НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ

«На правах рукопису»
УДК _____

«До захисту допущено»
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)
“___” _____ 2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 113, Прикладна математика _____
(код і назва)

на тему: Уточнення методу пошуку неможливих диференціалів для “Калина”-подібних шифрів _____
_____
_____

Виконав: студент __2__ курсу, групи ___ФІ-73мп_____
(шифр групи)

Турчин Андрій Ярославович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: доцент, к.т.н Яковлєв С.В. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»
Фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти: другий (магістерський) за освітньо–професійною програмою

Спеціальність: 113 «Прикладна математика»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)
«___»_____201_ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Турчин Андрій Ярославович _____
(прізвище, ім'я, по батькові)

1. Тема дисертації Уточнення методу пошуку неможливих диференціалів для "Калина"-подібних шифрів
_____
_____
_____ ,
науковий керівник дисертації Яковлєв С.В. доцент, к.т.н _____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від _____р. № _____

2. Термін подання студентом дисертації _____

3. Об'єкт дослідження процеси криптографічного перетворення інформації та методи їх аналізу _____
_____

4. Предмет дослідження (Вхідні дані – для магістерської дисертації за освітньо–професійною програмою)
Криптографічні властивості "Калина"-подібних схем шифрування _____
_____
_____

5. Перелік завдань, які потрібно розробити _____
_____
_____

_____
_____
_____

6. Орієнтовний перелік ілюстративного матеріалу _____
_____
_____

7. Орієнтовний перелік публікацій _____
_____
_____

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника магістерської дисертації.

РЕФЕРАТ

Кваліфікаційна робота містить: 80 стор., 11 рисунків, 2 таблиці, 8 джерел.

Метою дослідження є аналіз та уточнення методів криптографічного аналізу блокових шифрів.

Для досягнення мети необхідно розв'язати **задачу дослідження**, яка полягає у вдосконаленні методів пошуку неможливих, орієнтованих на «Калина»-подібні шифри, що дозволить, зокрема, уточнити оцінки стійкості національного державного стандарту України «Калина» до аналізу неможливих диференціалів.

Об'єктом дослідження є процеси криптографічного перетворення інформації та методи їх аналізу.

Предметом дослідження є криптографічні властивості «Калина»-подібних схем шифрування.

У ході роботи було виконано:

1) аналіз та модифікацію існуючого методу автоматизованого пошуку неможливих диференціалів для блокових шифрів;

2) формалізацію отриманих алгоритмів для «Калина»-подібних шифрів;

3) програмну реалізацію отриманих алгоритмів;

4) по отриманим даним проведений повторний аналіз стійкості «Калина»-подібних шифрів.

БЛОКОВІ ШИФРИ, АНАЛІЗ НЕМОЖЛИВИХ ДИФЕРЕНЦІАЛІВ, ШИФР «КАЛИНА»

РЕФЕРАТ

Квалификационная работа содержит: 80 стр., 11 рисунков, 2 таблицы, 8 источников.

Целью иследования является анализ и уточнение методов криптографического анализа блочных шифров.

Для достижения цели необходимо решить **задачу исследования**, которая заключается в совершенствовании методов поиска невозможных, ориентированных на «Калина»-подобные шифры, что позволит, в частности, уточнить оценки устойчивости национального государственного стандарта Украины «Калина» к анализу невозможных дифференциалов.

Объектом иследования являются процессы криптографического преобразования информации и методы их анализа.

Предметом иследования является криптографические свойства « Калина » - подобных схем шифрования.

В процесе роботы было проделано такие задачи:

1) анализ и модификацию существующего метода автоматизированого поиска невозможных диференциалов для блочных шифров;

2) формализацию полученных алгоритмов для «Калина» - подобных шифров;

3) программную реализацию полученных алгоритмов;

4) по полученным данным проведен повторный анализ устойчивости «Калина»-подобных шифров.

БЛОЧНЫЕ ШИФРЫ, АНАЛИЗ НЕВОЗМОЖНЫХ ДИФФЕРЕНЦИАЛОВ, ШИФР «КАЛИНА»

ABSTRACT

Qualifying work includes: 80 pg., 11 pictures, 2 tables, 8 sources.

The research task – improve and specify methods of impossible differential cryptoanalisys for block and oriented for «Kalyna»-like cyphers. It helps us make more correct estimation of DSTU «Kalyna».

The Object of research is the cryptographic transformations of information and methods of its analisys.

The Subject of research is cryptographic properties ща «Kalyna»-like cyphers.

The main points of this project:

1) analisys and modification of existing method of automatic search of truncated impossible differentials for block cyphers;

2) formalization of these methods for «Kalyna»-like cyphers;

3) program realization of these methods;

4) final analisys of results

BLOCK CYPHERS, IMPOSSIBLE DIFFERENTIAL ANALISYS, «KALYNA» CYPHER

The goal of research: make analisys and specify methods of cryptoanalisys of block cyphers

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ФТІ — Фізико-технічний інститут

$\oplus$ — операція побітового додавання;

НД — неможливий диференціал;

ВТ — відкритий текст;

ШТ — шифртекст;

$SubBytes$ — підстановка байтів;

$ShiftRows$ — зсув рядків;

$MixColumns$ — перемішування стовпців;

$AddKey$ — додавання ключа.

$\nrightarrow$ – неможливий перехід

ВСТУП

Актуальність дослідження. Метод аналізу неможливих диференціалів являється доволі зручним та ефективним способом оцінки стійкості шифрів. Окрім цього, даний метод має достатньо потенційних областей покращення та розширення. Саме тому, даний напрямок являється настільки цікавим та потенційно корисним напрямком всеможливих досліджень.

Метою дослідження є аналіз та уточнення методів криптографічного аналізу блокових шифрів. **Задачею дослідження** є вдосконалення методів пошуку неможливих, орієнтованих на «Калина»-подібні шифри, що дозволить, зокрема, уточнити оцінки стійкості національного державного стандарту України «Калина» до аналізу неможливих диференціалів. Для цього вирішувались такі задачі:

1) був проведений аналіз опублікованих по даній темі статей та методів;

2) було проведено дослідження на предмет доцільності доповнення алгоритму для «Калина»-подібних шифрів, який міг би дати покращені результати;

3) доповнити вже реалізований метод пошуку НД для «Калина»-подібних шифрів;

4) Оцінити стійкость шифру «Калина» до аналізу неможливих диференціалів із застосуванням запропонованого методу.

Об'єктом дослідження є процеси криптографічного перетворення інформації та методи їх аналізу.

Предметом дослідження є криптографічні властивості «Калина»-подібних схем шифрування.

При розв'язанні поставлених завдань використовувались такі *методи дослідження*: математичного моделювання, дискретної математики, комбінаторного аналізу, теорії імовірностей та математичної статистики,

лінійної алгебри.

Наукова новизна отриманих результатів полягає в тому, що не існує формальної теорії стійкості для для такої атаки як аналіз неможливих диференціалів. Саме тому дослідження шифрів і схем шифрування на стійкість до нього є важливим аспектом. В роботі для низки схем шифрування одержано умови, при виконанні яких дані шифри можуть вважатись практично стійкими до аналізу методом неможливих диференціалів.

Практичне значення результатів полягає в тому, що використовуючи результати аналізу неможливих диференціалів, можна уточнити оцінки стійкості схеми шифрування для неуможливлення потенційних атак.

1 ПОНЯТТЯ НЕМОЖЛИВОГО ДИФЕРЕНЦІАЛУ ТА РОЗВИТОК МЕТОДІВ ЇХ ПОШУКУ

Практично за останні два останні десятиліття з однаковою регулярністю виходять все нові наукові статті з описом нових способів пошуку НД. В більшості ідея та суть висунутих методів базується на попередніх самих ефективних методах. Нові методи завжди покривають результати своїх попередників і вдосконалюють їх або ж в швидкодії, або ж новими результатами.

Для опису висунутих ідей в наступних розділах досить доцільним буде провести екскурс по основоположним методам в даному напрямку. Це допоможе плавно ввести в дану роботу більшість ідей, що стоятимуть за побудовою готового алгоритму.

Оскільки дане дослідження продовжує мою попередню роботу в даному напрямку, то доцільним буде вказати здобуті результати, які я отримав перед цим в кінці даного розділу.

1.1 Перші праці, де розглядались неможливі диференціали

Сама ідея досліджувати диференціали з ймовірністю 0 була вперше запропонована Ларсом Кнудсеном для аналізу DEAL в 1998 році і розвинена Елі Біхамом для атаки на IDEA та Skipjack. Для кращого розуміння сутності того, що з себе представляють неможливі диференціали, розглянемо деякі викладки з роботи Біхама і Келлера "Cryptanalysis of Reduced Variants of Rijndael" [4].

Як відомо, принцип роботи шифру AES наступний:

– довжина блоку ВТ (ШТ відповідно) може приймати значення 128,

192 та 256 бітів;

– довжина ключа може приймати значення 128, 192 та 256 бітів;

– кількість циклів приймає значення від 10 до 14 в залежності від перших двох параметрів.

Блок ВТ, а також ключ зручно представляти записаними у таблиці з чотирма рядками та N_b і N_k стовпцями відповідно ($N_b, N_k \in \{4, 6, 8\}$):

a_{00}	a_{01}	...
a_{10}	a_{11}	...
a_{20}	a_{21}	...
a_{30}	a_{31}	...

(а)

k_{00}	k_{01}	...
k_{10}	k_{11}	...
k_{20}	k_{21}	...
k_{30}	k_{31}	...

(б)

Рисунок 1.1 – Зображення блоку ВТ та ключа у вигляді таблиць: (а) - блок ВТ N_b, (б) - ключ N_k

У кожній клітинці таблиці записаний один байт. Таблиці заповнюються зверху до низу по стовпцях. Один стовпчик таблиці є 4-байтовим словом. N_b і N_k – кількість слів у відповідній таблиці. Наприклад, якщо довжина блоку відкритого тексту = 192, то $N_b = 6$ і т.д. Параметри N_b і N_k вибираються незалежно один від одного.

Кількість циклів N_r визначається виходячи з вибраних довжин блоку ВТ і ключа згідно таблиці

		Nk		
		4	6	8
Nb	4	10	12	14
	6	12	12	14
	8	14	14	14

Рисунок 1.2 – Таблиця відповідностей N_r від (N_b, N_k)

Кожен проміжний результат перетворень, що виконуються у криптоалгоритмі, будемо називати станом. Стан на кожному кроці алгоритму також будемо зображати у вигляді таблиці з N_b слів.

Перетворення на одному циклі :

1) $SubBytes$ (підстановка байтів);
2) $ShiftRows$ (зсув рядків);
3) $MixColumns$ (перемішування стовпців);
4) $AddKey$ (додавання ключа).

Причому останній цикл не містить етапу перемішування стовпців.

The Square Attack

Досить тривалий проміжок часу найбільш результативною атакою на AES вважався “ The Square Attack” - метод, що був висунутий власне самими ж авторами.

Якщо вдаватись в деталі, власне ідея атаки будувалась на факті того, що якщо розглядати версію шифру AES в чотири раунди, де на вході вказані 256 наборів ВТ, що мають всі можливі відмінні значення в одному конкретному байті, а в усіх решту відрізняється, то на вході перетворення $SubBytes$ четвертого раунду XOR всіх станів в кожному байті буде рівним нулю. Ця властивість диктується структурою самого шифру.

Таким чином для атаки на нього вдавались до наступної послідовності кроків. Підбирались 256 наборів ВТ, де перший байт кожного з них був відмінного від решту усіх значення. Всі інші байти, що залишались, для кожного з наборів були однакові. Далі вгадувався єдиний невідомий байт у раундовому ключі номер чотири, і з врахуванням його значення отримувався відповідний стан перед $SubBytes$ перетворенням четвертого раунду. Тоді, згідно з властивістю, що була надана в попередньому абзаці, якщо XOR всіх станів в кожному байті буде рівним нулю, то нехвataючий байт раундового ключа був підібраний правильно. Якщо ні, то робилось нове припущення, щодо значення відповідного байту раундового ключа.

Сам же спосіб знаходження неймовірних диференціалів для цієї

конфігурації шифру, виник як спосіб отримання кращих результатів для описаної вище атаки.

Покращення The Square Attack, з використанням неможливих диференціалів

При відмінності ВТ тільки в одному байті відповідні пари ШТ нізащо не будуть одинаковими в наступних комбінацях байтів:

– (1, 6, 11, 16);

– (2, 7, 12, 13);

– (3, 8, 9, 14);

– (4, 5, 10, 15)

Це пояснюється "розмазуванням"різниць станів після двох раундів. А саме, якщо різниця станів перед першою операцією $MixColumns$ є в одному байті, отож після неї різниця буде поширена по одному стовпцю, а вже опісля операції $MixColumns$ другого раунду стани відрізнятимуться в усіх байтах.

З іншої сторони, якщо ШТ на виході буде рівним одній з "заборонених"комбінацій, що були описані двома абзацами вище, значить після третьої операції $MixColumns$ дані будуть рівними в одному стовпці. Це в свою чергу означає, що перед другою операцією $MixColumns$ є чотири байти з однаковими значеннями. Отримуємо протиріччя, в силу того, що раніше ми стверджували про різницю всіх байтів на даному етапі. Наглядніше зрозуміло суть описаних речей на рисунку 1.3.

Слід зауважити про деякі спрощення в даній моделі. Перетворення, що не вносили ніяких змін на проміжні стани, такі як $SubBytes$ та $AddKey$, не розглядались.

Цей доволі простий приклад є дуже наглядним для описання суті НД. Якщо підсумувати її в розрізі даного прикладу, то оперуючи перетворенням пар різниць байтів здобувались деякі проміжні результати з ймовірністю 1. Пропускаючи набори різниць через перетворення по обидві сторони шифру, ми маємо можливість отримати протиріччя на їх

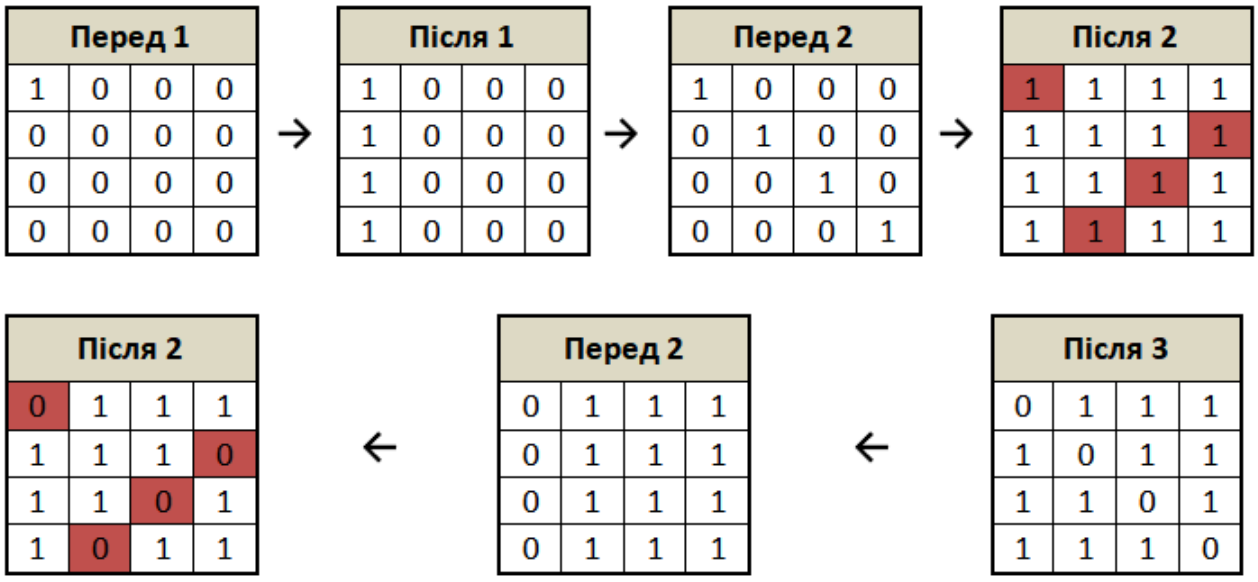


Рисунок 1.3 – Приклад існування НД

«стиці». Зрозуміло ж що два стани, які отримані з ймовірністю 1 незалежно один від одного, не можуть мати протиріччя.

Даний метод застосовується для припущення щодо вигляду, і в результаті поступового відкидання множини підключів. Дана процедура зазвичай проводиться ітеративно поки не отримуємо множину власне ключів, що можна відкидати.

Звісно, в даному методі орієнтуються тільки на специфіку роботи AES і він не є застосовним для блокових шифрів з іншою архітектурою. Але все ж він був досить важливим для розвитку способів пошуку НД в даному напрямку в свій час.

1.2 Узагальнений криптоаналіз НД на структурах блокових шифрів

Як і було зазначено, описаний вище метод є досить наглядним для описання сутності НД і способів їх пошуку, але він зовсім не несе

узагальнений характер. З часом було розроблено багато схожих «спеціальних» методів для різних шифрів, які давали досить хороші результати, але теж носили частковий характер. Як можна догадатись вони були просто чітко продумані під архітектуру якогось конкретного шифру. Звісно вони могли давати досить серйозні результати, але як серйозний спосіб дослідження стійкості шифрів в майбутньому не розглядались.

Все ж згодом був висунутий досить вдало формалізований метод, що міг описати і врахувати особливості більшості блокових шифрів чи скажімо легко модифікувався в залежності від шифру. Такий метод можна було вже задіяти для автоматизації обчислень, легшого отримання результатів і порівняння стійкості шифрів між собою. В літературі він відомий як UID-метод (Unified Impossible Differential Cryptoanalys)[2].

Процес, що лежить в основі атак за допомогою НД в даному методі складається з таких основних кроків:

– описання всіх можливих типів значень, що можуть набувати змінні, потрібних для роботи;

– задання всіх можливих наборів варіантів типів вхідних та вихідних станів;

– формальне задання усіх шифруючих перетворень між його різними станами для роботи з ними;

– вибрати вхідні та вихідні стани з множини можливих;

– ітеративно пропускати стани в двох *незалежних потоках: ініціалізованих вхідним та вихідним станом відповідно;

– знаходження невідповідностей на проміжних раундах (на «стиці» двох потоків);

Іншими словами при переборі всіх можливих пар "вхід-вихід"шифру, знаходяться пари характеристик з ймовірністю 1, що суперечать одна одній. Потім відкидаються всі пари підключів, які відповідають цій множині.

Для формальності під неможливою диференційною характеристикою

довжини r, будемо підрозумівати структуру, яку позначатимемо наступним чином:

$(x_1, x_2, \ldots, x_n) \nrightarrow_r (y_1, y_2, \ldots, y_n)$,

що означає що після r раундів вхідна різниця $(x_1, x_2, ..., x_n)$ не може перейти в вихідну $(y_1, y_2, ..., y_n)$.

Нехай даний блочний шифр S, з n підблоками. Якщо вхідна різниця має вигляд $U = (u_1, \ldots, u_n)$, тоді назвемо U – вектором різниці і u_i, $0 \leq i \leq n$ – різниця i-го під-блоку. Позначимо вихідну різницю для U після r раундів U^r і позначимо значення i-го підблоку для різниці U^r як U_i^r. Отримана вихідна різниця кожного підблоку після r раундів є лінійною XOR комбінацією наступних чотирьох типів різниць: Нульова різниця – різниця яка дорівнює нулю і позначається 0. Ненульова фіксована різниця – різниця, яка не рівна нулю і приймає фіксоване значення, позначимо її l_i. Ненульова змінна різниця – різниця, яка приймає будь-яке відмінне від нуля значення, позначимо її m_i. Змінна різниця – різниця, яка приймає будь-яке значення, позначимо її r_i. Серед цих чотирьох типів різниць, ненульова фіксована різниця l_i і ненульова змінна різниця m_i не можуть бути рівними 0, а змінна різниця r_i може бути рівна як нульовій різниці, так і ненульовій різниці.

Введемо наступне означення: два вектори $U = (U_1, U_2, ..., U_n)$ і $V = (V_1, V_2, ..., V_n)$ називаються несумісними, якщо існує підпослідовність $I \subset \{1, 2, \ldots, n\}$ така, що XOR різниць в підпослідовності завжди не є рівним:

$$\oplus_{i \in I} U_i \neq \oplus_{i \in I} V_i \tag{1.1}$$

Наприклад, якщо $U = (l_1 \oplus m_1, 0)$ і $V = (l_1, 0)$, де l_1 є ненульовою фіксованою різницею і m_1 є ненульовою змінною різницею, тоді U і V є несумісними, оскільки $l_1 \oplus m_1$ не може бути рівним l_1.

Якщо $U = (u_1, u_2) = (l_1, l_1 \oplus m_1)$ і $V = (v_1, v_2) = (m_1, m_2)$, тоді $u_1 \oplus u_2 = m_1$ і $v_1 \oplus v_2 = 0$, завжди є не рівними таким чином U і V є

несумісними.

Нехай для блочного шифру S даний вхідний вектор U^0 і вихідний вектор V^0 ми можемо порахувати вектор різниці U^i через i раундів шифрування і аналогічно порахувати вектор V^j через j раундів розшифрування. Якщо знайдені вектори є несумісними згідно з 1.1, то значить існує $i + j$ – раундовий неможливий диференціал $U^0 \nrightarrow_{i+j} V^0$.

Ми вважаємо, що маємо три різних види перетворень в блочних шифрах: нульове перетворення 0, ідентичне перетворення 1 і нелінійне перетворення F. Якщо вхідна різниця дорівнює 0, тоді після F перетворення вихідна різниця також буде дорівнювати 0. Якщо вхідна різниця рівна ненульовій фіксованій різниці l_i, тоді після F трансформації вона буде дорівнювати m_j, новій ненульовій змінній різниці. В іншому випадку, вихідною різницею буде r_j, що являється новою змінною різницею . Дані три перетворення наведені нижче у таблиці 1.1.

Таблиця 1.1 – Типи перетворень у блочних шифрах

Перетворення	Вхід	Вихід	Відмітки
0	$x \in \{0, l_i, m_i, r_i\}$	0	Нульове перетворення
1	$x \in \{0, l_i, m_i, r_i\}$	x	Ідентичне перетворення
F	0	0	F – перетворення
	l_i	m_j	
	m_i	m_j	
	Інакше	r_j	

Пошук неможливих диференціалів для матричного представлення

Всі перетворення в блочному шифрі можливо задати матрицею переходу. Нехай ми отримали для деякого шифру відповідні матриці шифрування та дешифрування. Тоді результат множення вектору різниць $U = (u_1, \ldots, u_n)$ на характеристичну матрицю шифрування(дешифрування) $E(D)$ розуміється як використання перетворення e_{in} до різниці u_i.

Наприклад, якщо вхідна різниця для фейстелівської схеми буде рівною $U = (l_1, 0)$, тоді

$$U \cdot E = \Big(l_1, 0\Big) \cdot \begin{pmatrix} 0 & 1 \\ & \\ 1 & F \end{pmatrix} = \Big(l_1, 0\Big) \tag{1.2}$$

Існують деякі блочні шифри, які не можуть бути перетворені в UID-форми, що наведене вище. В такому випадку, ми можемо перетворити їх в композицію декількох UID-форм, і для кожної форми буде існувати відповідна характеристична матриця.

Отже, в першу чергу для потрібної схеми шифрування необхідно побудувати відповідну матрицю перетворень станів. За допомогою неї можна буде отримувати вигляд станів через необхідну кількість раундових перетворень.

Нехай в нас є стан блоку через i раундів $U^i = ((U^0 \cdot E) \cdot \dots \cdot E)$, який отриманий поступовим множенням на матрицю перетворення i разів, а також вектор різниці через j раундів дешифрування, отриманий з вихідного вектору V^0: $V^i = ((V^0 \cdot D) \cdot \dots \cdot D)$. Якщо U^i та V^j є несумісними, то це означає, що ми знайшли неможливий диференціал довжини $i + j$ такого вигляду: $U^0 \nrightarrow_{i+j} V^0$

Вцілому, даний метод давав хороші результати. Але все ж багато «спеціалізованих» методів давали кращі результати, тобто знаходила НД більшої довжини. Це означало, що багато важливої інформації все ж втрачалось по ходу роботи алгоритма.

1.3 Загальний опис методу У-Ваня

У 2012 році У-Ванем було висунуто доволі відмінний від усіх попередніх способів знаходження неможливих диференціалів. Як стверджувалось, метод був узагальненим випадком UID-методу, а тому не втрачав своєї загальності.

Головним плюсом свого методу У-Вань висували те, що на відміну від UID-методу, який використовує так званий $Miss - in - the - middle$ підхід, на їхній метод не діють дані обмеження. Тобто, алгоритм не націлений, щоб на «стиці» або ж знайти протиріччя, або ж ні і завершити розгляд даної пари вхід-вихід. Натомість, він здобуває нову інформацію і продовжує роботу, де або знаходить протиріччя, або ж, дійсно, завершає роботу, якщо нової інформації здобути вже неможливо.

Згідно з результатами, вказаними в даній статті, даний метод окрім того, що повторив найкращі результати UID-методу, він ще й повторив результати багатьох ефективних спеціалізованих неузагальнених методів, які давали кращі результати за UID-метод, але були розроблені конкретно під котрусь з схем шифрування.

Довжина неможливих диференціалів нас цікавить з розрахунку на те, що чим довший неможливий диференціал знайдений, тим кращу атаку на шифр ми можемо побудувати.

Отож, опиратимемось на метод автоматичного пошуку неможливих диференціалів, який може бути використаним для словоорієнтованих блокових шифрів з бієктивними S-блоками, який об'єднав в собі особливості UID- методу та багатьох відомих спеціальних методів.

Входом для даного алгоритму служать деякі обмеження на вигляд вхіного та вихідного стану і система рівнянь, що описує проходження станів через внутрішні перетворення шифру. Після цього алгоритм передбачає значення змінних стану з ймовірністю 1 на кожному кроці. В

результаті, алгоритм видає можливість існування неможливого диференціалу з розрахунку інформації, яку ми йому надали.

Одним з його основних блоків вважається побудова системи рівнянь, що описує поведінку різниць при проходженні певних внутрішніх перетворень шифра. Нехай $\Delta X = (\Delta x_i)_{1 \leq x_i \leq n}$, $\Delta Y = (\Delta y_i)_{1 \leq y_i \leq n}$, $\Delta Z = (\Delta z_i)_{1 \leq z_i \leq n}$ є векторами над F_2^n.

Вважатимемо, що надалі в даній роботі ми розглядатимемо блокові шифри довжини $l \cdot s$ біт, де s-бітовий розмір машинного слова. Також, як і раніше, перетворення $AddKey$ не впливатиме ніяк на дану модель, тому воно опускатиметься.

Для початку варто описати основні з можливих перетворень, які дуже часто присутні в більшості відомих шифрів:

1) Для **операції гілкування** ми маємо $\Delta X = \Delta Y = \Delta Z$. Це рівнняння може бути описане як $2n$ лінійних рівнянь $\Delta x_i \oplus \Delta y_i = 0$ та $\Delta x_i \oplus \Delta z_i = 0$

2) Для **XOR-операції** ми маємо $\Delta X \oplus \Delta Y = \Delta Z$. Це рівнняння може бути описане як n лінійних рівнянь $\Delta x_i \oplus \Delta y_i \oplus \Delta z_i = 0$

3) Для **операції лінійного перемішування** ми маємо $\Delta Y^T = P \cdot \Delta X^T$, де P є матричним представленням, що задає лінійне перемішування. Це рівнняння може бути описане як n лінійних рівнянь $\Delta y_i \oplus \bigoplus_{j=1}^{n} p_{i,j} \cdot \Delta x_i = 0$

4)Для **шару S-блоків з бієктивним відображенням** $S_i : F_2^s \to F_2^s$, будується n рівнянь виду $\overline{S_i}(\Delta x_i, \Delta y_i) = 0$

Користуючись формальним описом кожного з перетворень можна скласти систему рівнянь, яка відповідатиме двом найважливішим та найпоширенішим класам блокових шифрів, а саме SP-мережам та схемам фейстелівського типу.

SP-мережі. Один раунд шифрування SP-мережі, як правило, складається з трьох шарів:

– шар додавання раундового ключа;

– шар S-блоків;

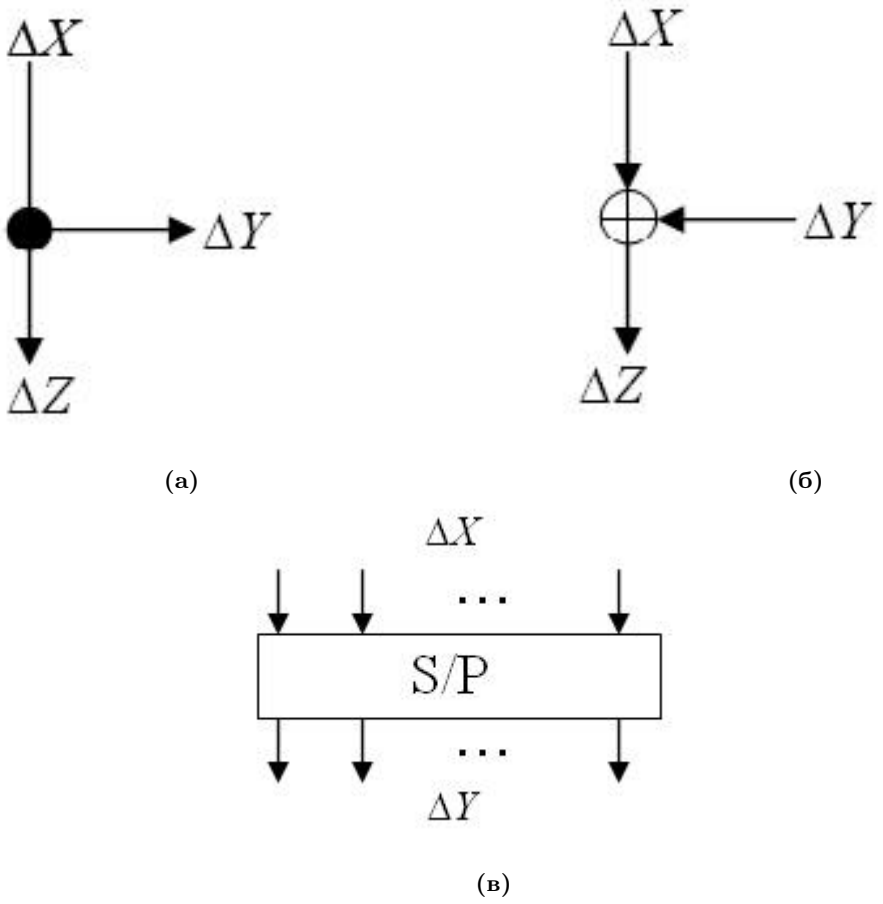
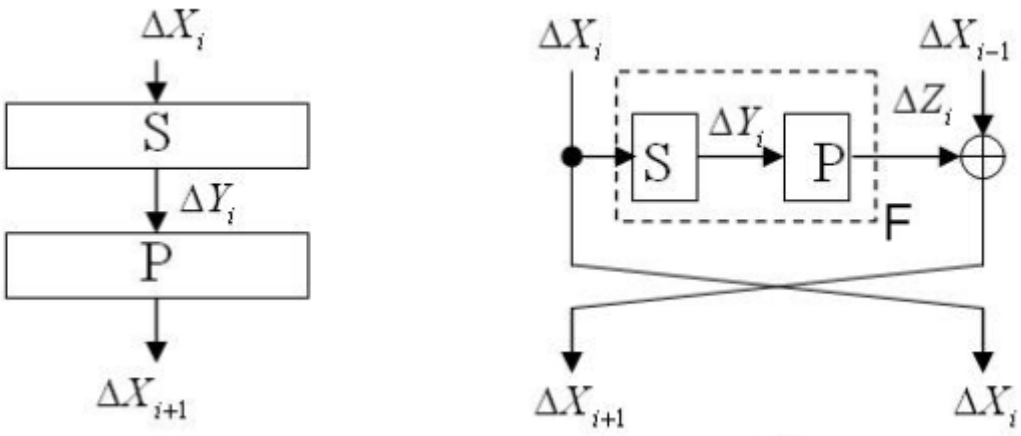


Рисунок 1.4 – Основні перетворення у блокових шифрах: (а) операція гілкування, (б) XOR-операції, (в) операція лінійного перемішування / шар S-блоків.

– шар лінійного перемішування.

Як і в розглянутих раніше методах, шар додавання раундового ключа не вносить ніяких змін в вид нашої моделі, тому ми його не враховуємо. Також останній шар лінійного перемішування не впливає на довжину результуючого неможливого диференціалу.

Вважатимемо, що $\Delta X = (\Delta x_i)_{1 \le x_i \le n}$, $\Delta Y = (\Delta y_i)_{1 \le y_i \le n}$ є різницями перед і після шару S-блоків відповідно. Тоді відповідні системи, описуючі проходження станів через внутрішні перетворення даного типу шифріву буде мати наступний вигляд:



(а) **(б)**

Рисунок 1.5 – Схеми найважливіших класів блокових шифрів: (а) SP-мережі, (б) шифрів фейстелівського типу.

$$\begin{cases} \overline{S_i}(\Delta X_{i,j}, \Delta Y_{i,j}) = 0, & 1 \le x \le r, 1 \le j \le l \\ \Delta X_{i+1}^T \oplus P \cdot \Delta Y_i^T = 0, & 1 \le x \le r-1 \end{cases}$$

Шифри фейстелівського типу. Для r-раундового шифру Фейселівського типу вважатимемо, що $\Delta X_{i-1} = (\Delta X_{i-1,j})_{1 \le j \le l/2}$ і $\Delta X_i = (\Delta X_{i,j})_{1 \le j \le l/2}$ є різницями правої та лівої гілки для i-ого раунду шифрування відповідно. Як і для SP-мережі, раунд шифрування для шифрів Фейстелівського типу складається з тих же шарів перетворень.

Позначимо також, що $\Delta Y_i = (\Delta Y_{i,j})_{1 \le j \le l/2}$ – різниця станів після шару S-блоків i-ого раунду шифрування і $\Delta Z_i = (\Delta Z_{i,j})_{1 \le j \le l/2}$ – різниця станів після F-функції i-ого раунду шифрування. Тоді відповідні системи, описуючі проходження станів через внутрішні перетворення даного типу шифрів будуть мати наступний вигляд:

$$\begin{cases} \overline{S_i}(\Delta X_{i,j}, \Delta Y_{i,j}) = 0, & 1 \le x \le r, 1 \le j \le l/2 \\ \Delta Z_i^T \oplus P \cdot \Delta Y_i^T = 0, & 1 \le x \le r \\ \Delta X_{i-1} \oplus \Delta X_{i-1} \oplus \Delta Z_i = 0, & 1 \le x \le r \end{cases}$$

,де P – матриця лінійного перетворення шару F.

На прикладі побудови систем рівнянь для даних двох типів шифрувань, можна побудувати більш відповідну і ефективну систему для інтересуючого нас шифру.

1.4 Результати попередніх досліджень

Попередня робота в даному напрямку була висунута з основною ідеєю адаптування та допрацювання основних ідей, описаних в методі У-Ваня стосовно «Калина»-подібних шифрів, розробка програмної реалізації висунутого нового алгоритму та отримання результатів стосовно стійкості даного класу шифрів перед атаками з застосуванням аналізу неможливих диференціалів.

Іншими словами було попередньо проаналізовано основні техніки та напрацювання в роботі У-Ваня. З іншої сторони, було проаналізовано структуру шифру «Калина-2» та його можливих модифікацій в термінах роботи У-Ваня. Використавши все вищеперечислене, було запропоновано новий, адаптований саме під «Калина»-подібні шифри метод. Він в свою чергу був доповнений деякими новими правилами переходів.

Запропонована методика була застосована до зменшеної версії шифру «Калина» із блоком у 64 біти та до оригінальних версії шифрів «Калина-128» та «Калина-256». Даний метод знайшов широкий клас трираундових НД для різних модифікацій шифру «Калина». Також отримані результати, дали повід висунути допущення, що при певних допрацюваннях можливо знайти було ще ряд класів НД.

Сама можливість існування більшого класу неможливих диференціалів, які можна знайти за допомогою ускладнення даного методу випливає з можливості пропускання деякої кількості проміжної інформації між застосуванням формалізованих правил. Таким чином, в моделі потенційно існували частини, які давали не достатньо зворотньої інформації, яку можна було б використовувати для отримання протиріч. Одним з таких вузьких місць даного методу була відсутність кроку розв'язування системи лінійних рівнянь на кожному з раундів. Реалізувавши цей функціонал, метод одразу ж почне підтримувати

набагато ширший клас «Калина»-подібних шифрів. Також вже після отримання результатів була опублікована ще одна робота в даному напрямку, яка пропонувала деякі цікаві уточнення до методу У-Ваня та несла більш загальний характер.

Висновки до розділу 1

Даний розділ слугує коротким екскурсом в предметну область даної роботи. Розділ починається з описання вироджених випадків, де чітко прослідковується перші випадки застосування поняття НД, для практичних цілей.

Поступово розглядаються самі вагомі методи в області зі своїми деталями роботи. Дані методи необхідно було описати, ввиду перенесення багатьох цих ідей в "фінальний"метод в даній роботі, опис якого почнеться починаючи з наступного розділа.

2 ФОРМАЛІЗАЦІЯ ПРАВИЛ ДЛЯ ПОБУДОВИ УТОЧНЕНОГО МЕТОДУ

В даному розділі детально розглядаються всі необхідні теоретичні та практичні напрацювання для побудови вже власне вдосконаленого методу пошуку неможливих диференціалів для «Калина»-подібних шифрів. Почавши з не складного математичного апарату, що використовуватиметься в певній частині методу, буде описано метод У-Ваня, що є основоположним для всього "нового"алгоритму. Для кращого його розуміння буде описано характерну різницю між методом У-Ваня та UID-метода.

З іншої сторони необхідно буде коротко описати основні деталі шифру «Калина-2» та його внутрішніх перетворень.

Під кінець розділу будуть подані основні тезиси для побудови формальних правил та реалізації алгоритму згідно з ними.

2.1 Розвязуваність лінійних систем

Важливим доповненням до попередньої множини правил, що вносились до правил у методі У-Ваня стосовно «Калина»-подібних шифрів, є здобуття додаткової інформації за допомогою винесення додаткової інформації на сувмісність системи лінійних рівнянь на кожному раунді. Отже, перед тим як приступити до детального опису подальших методів пошуку неможливих диференціалів потрібно розглянути основи лінійної алгебри, що визначають вирішуваність системи лінійних рівнянь.

Нехай m та n будуть позитивними цілими числами такими, що $m < n$,

і $Ax = b$ є системою m лінійних рівнянь з n змінними, де A – матриця розмірності $m \cdot n$ над полем F_2 і $x = (x_1, x_2, ..., x_n)$ та $b = (b_1, b_2, ..., b_n)$ є двійковими векторами, тоді доповнена $m \cdot (n + 1)$ матриця $B = [A|b]$ буде визначати вирішуваність системи.

Одним з поширених способів є виведення зменшеної ешелонної форми рядка матриці за допомогою алгоритму Гауса-Жордана розв'язання систем лінійних алгебраїчних рівнянь. Цей спосіб використовувався в ряді робіт по даному напрямку.

В загальному випадку цей метод працює за наступним алгоритмом:

– Обирається перша зліва колонка, що містить хоч одне ненульове значення;

– Якщо верхнє число у цій колонці - нуль, то обмінюється увесь перший рядок матриці з іншим рядком матриці, де у цій колонці нема нуля;

– Усі елементи першого рядка діляться на верхній елемент обраної колонки;

– Від рядків, що залишились, віднімається перший рядок, помножений на перший елемент відповідного рядка, з метою отримання нуля в першому елементі кожного рядка (крім першого);

– Далі, повторюємо ці операції із матрицею, отриманою з початкової матриці після викреслювання першого рядка та першого стовпчика

– Після повторення операцій $n-1$ разів отримаємо верхню трикутну матрицю;

– Віднімаємо від передостаннього рядка останній рядок, помножений на відповідний коефіцієнт, щоб у передостанньому рядку залишилась лише 1 на головній діагоналі;

– Повторюємо попередній крок для наступних рядків. У результаті отримуємо одиничну матрицю і рішення на місці вільного вектора (над ним необхідно виконувати ті самі перетворення);

Зменшена форма ешелону рядка матриці унікальна і позначається як B'. Один починає перевіряти B' від останнього ряду до першого, щоб

побачити, чи існує рядок в якому перші n записів - нулі, а останній запис - ненульовим. Якщо є такі рядки, то лінійна система не має вирішення. Наприклад, якщо доповнена матриця B лінійної системи в зменшеному ряду ешелонова форма є наступною:

$$B' = \begin{pmatrix} 1, 0, 0, b_1 \\ 0, 1, 0, b_2 \\ 0, 0, 0, b_3 \end{pmatrix} \tag{2.1}$$

, де b_3 є ненульовим , лінійна система рівнянь немає розв'язків.

З іншої сторони, як буде видно нижче, в наслідок обрання специфічної підмножини множини всіх можливих пар входів-виходів для шифрів даного класу дуже поширеною буде ситуація так званого "викреслення"стовпців та рядків матриці на більшості раундів. Ця ситуація отримується як наслідок розгляданих нульових типів змінних на вході і виході перетворення. В такому разі розглядаєма матриця буде відмінною в залежності від пар входів-виходів для даного перетворення.

В такому разі, зручним способом для вияснення сумісності системи лінійних рівнянь на кожному етапі стане застосування теореми Кронекера-Капеллі. В ній сказано:

Система лінійних алгебраїчних рівнянь являється сумісною тоді і тільки тоді, коли ранг матриці системи є рівним рангу розширеної матриці системи $rangB \neq rangB'$. Звідси, як наслідок випливає що:

– Якщо $rangB \neq rangB'$, то система лінійних алгебраїчних рівнянь є несумісною;

– Якщо $rangB = rangB' < n$, то система лінійних алгебраїчних рівнянь є невизначеною;

– Якщо $rangB = rangB' = n$, то система лінійних алгебраїчних рівнянь є визначеною;

2.2 Метод У-Ваня, як базовий

Головна ідея, що використовується в цьому методі для знаходження неможливих диференціалів є досить простою: В першу чергу, фіксується певна інформація про різниці вхідного та вихідного тексту. За допомогою попередньо складених формальних правил, згідно до архітектури шифру що розглядається, можемо передбачити інформацію про нові змінні і таким чином отримати новий набір відомих змінних. не знайдуться суперечності. Це означатиме вже існування неможливого диференціалу. Якщо ж ми більше не зможемо отримати будь-яку нову інформацію і нових ітерацій вже не передбачається, то НД не знайдено.

Вважатимемо, що для потрібної схеми шифрування внутрішні раундові перетворення можна умовно поділити на дві підсистеми: лінійну – L та нелінійну – NL.

L включає в себе всі лінійні перетворення, в той час коли NL відповідатиме за всі бієктивні S-блоки. Тоді нова інформація здобувається двома шляхами:

1)Здобути нову інформацію з лінійної частини:

Якщо L має розв'язок, тоді можна її рішити за допомогою метода Гауса-Жордана, який дає рішення зведенням розширеної матриці L до діагональної форми, використовуючи елементарні маніпуляції з рядками матриці. Зведена матриця в результаті буде еквівалентною вхідній.

Припустимо, що L має розв'язок і доповнена матриця отримана, тоді:

– якщо в цій доповненій матриці можна знайти афінне рівняння одної змінної виду $\Delta X \oplus c = 0$, де c – константа, то можливо тільки, що $\Delta X = 0$, якщо $c = 0$ і $\Delta X \neq 0$, якщо $c \neq 0$;

– якщо в цій доповненій матриці можна знайти лінійне рівняння двох змінних виду $\Delta X \oplus \Delta Y = 0$, то можливо тільки, що $\Delta X \neq 0$, якщо $\Delta Y \neq 0$;

2)Здобути нову інформацію з нелінійної частини:

Допустимо ми маємо бієктивний S-блок, ΔX та ΔY - вхідна та вихідна різниці відповідно.

Якщо ΔP та ΔC – вхідна і вихідна різниці відповідно, то матимемо $\Delta P \nrightarrow \Delta C$, якщо справедлива одна з двох ситуацій:

– лінійна система L немає розв'язків;

– існують одночасно нульові та ненульові змінні;

Наведемо невеликий приклад для другої ситуації:

Приклад 2.1. Нехай маємо рівняння $\Delta Y \oplus \Delta Z = 0$ та $S(\Delta X, \Delta Y) = 0$, що належать системі поширення різниць. Ми знаємо, що $\Delta X = 0$ та $\Delta Z \neq 0$ з отриманої раніше інформації. Тоді для наступної ітерації ми знаємо, що $\Delta Y \neq 0$ з правил обробки лінійної частини. Але також ми знаємо, що $\Delta Y = 0$ з правил обробки нелінійної частини.

Отже, ми отримали протиріччя.

Algorithm 2.1 Псевдокод алгоритму

```
1:  for (кожної пари (ΔP, ΔC)) do
2:      індикатор = false;
3:      індекс = false;
4:      while індекс do
5:          if (система L немає жодного рішення) then
6:              індикатор = true;
7:              індекс = false;
8:          else
9:              Передбачити інформацію з L;
10:             Передбачити інформацію з NL;
11:
12:             if (не знайдено ніякої нової інформації) then
13:                 індекс = false;
14:             else
15:                 if (знайдені змінні, що одночасно з ймовірністю 1 нульові та ненульові) then
16:                     індикатор = true;
17:                     індекс = false;
18:                 end if
19:             end if
20:         end if
```

Реалізація алгоритму в загальному випадку

Отже, L-систему ми формально записуємо, як $A \cdot x = b$, де A – матриця коефіцієнтів, b – вектор констант. x – набір **всіх** змінних, що проходитимуть через внутрішні перетворення. Для зручності, можна виразити цю систему наступним чином:

$A_1 \cdot x_1 \oplus A_2 \cdot x_2 \oplus \cdots \oplus A_n \cdot x_n = b,$

Примітно, що якщо змінити в ній порядок змінних, то й матриця A повинна змінити свій вигляд для відповідності.

Приклад 2.2. Якщо приміняти даний підхід до шифрів фейстелівського типу, то враховуючи його будову 1.4, розглянену раніше отримаємо: $I_1 \cdot \Delta X_{i-1}^T \oplus I_2 \cdot \Delta X_{i+1}^T \oplus P \cdot \Delta X_{i+1}^T = 0, 1 \leq i \leq r$, де $I_1 = I_2 = I$ – одиничні матриці. Завважимо, що ΔZ_i виключено з системи для спрощення.

Далі ми сортуємо всі змінні для спрощеної системи як:

$x = [\Delta X_0, \Delta X_1, \Delta X_2, \ldots, \Delta X_r, \Delta X_{r+1}, \Delta Y_1, \Delta Y_2, \ldots, \Delta Y_r,]$

В результаті отримуємо $A \cdot x^T = 0,$

де

$$A = \begin{pmatrix} l_1, 0, I_2, 0, \ldots, 0, 0, 0, P, 0, \ldots, 0 \\ 0, l_1, 0, I_2, \ldots, 0, 0, 0, 0, P, \ldots, 0 \\ \ldots, \ldots, \ldots, \ldots, \ldots, \ldots, \ldots, \ldots, \ldots, \ldots \\ 0, 0, 0, 0, \ldots, I_1, 0, I_2, 0, 0, \ldots, P \end{pmatrix} \tag{2.2}$$

є комплексною блоковою матрицею з r рядками та $2r + 2$ стовпцями.

2.3 Порівняння методу У-Ваня з UID-методом

Твердження 2.1. *UID-метод є частковим випадком, методу У-Ваня*

Доведення.

1) По-перше, матричне представлення в UID-методі, приклад якого був наведений в формулі 1.2, є специфічною формою системи проходження станів.

2) По-друге, операції, що використовуються в UID-методі, визначені в таблиці 1.1, а також в формулах 1.1 та 1.2 є включеними в правила здобування нової інформації, що наведені в розділі 2.1.

3) Ну й на кінець, буде показано, що протиріччя, яке здобуває UID завжди може бути знайдене і методом У-Ваня.

Згідно з UID-методом знаємо, що для виявлення НД необхідно, щоб виконувалась умова формули 1.1. Але ми цього не можемо зробити в деяких випадках. Ми не можемо нічого стверджувати, якщо вираз містить змінні з невідомим значенням, або ж дві змінні з ненульовим змінним значенням. Також метод не видає точне рішення якщо він містить стан з ненульовим незмінним значенням та з ненульовим змінним значенням якихось двох або більше своїх змінних відповідно.

Таким чином, ми дійсно можемо знайти протиріччя, якщо стани в нас містять тільки одну змінну з ненульовим змінним або ж незмінним значеннями.

Натомість, в методі У-Ваня будуємо l лінійних рівнянь виду $\Delta u_i \oplus \Delta v_i = 0$, $1 \leq i \leq l$ в системі розповсюдження різниць з введенням потрібних внутрішніх змінних. Ці лінійні рівняння включаються в систему L, отже маємо:

– якщо $\oplus_{i \in I}(\Delta u_i \oplus \Delta v_i) = \Delta c$, де Δc – ненульова змінна різниця. Звідси, ненульова змінна різниця належатиме простору $\{\Delta u_i \oplus \Delta v_i = 0 :$

$1 \leq i \leq l\}$. В цьому випадку, ранг матриці L не буде рівним рангу доповненої матриці. Так ми отримуємо НД згідно з правилами здобування нової інформації з лінійної частини;

– якщо $\oplus_{i \in I}(\Delta u_i \oplus \Delta v_i) = \Delta a$, де Δa – ненульова незмінна різниця. З означення $a \neq 0$, але якщо система рішається, то $a = 0$. Таким чином, знову отримуємо НД згідно з правилами здобування нової інформації з лінійної частини.

Таким чином, UID дійсно є частковим випадком методу У-Ваня. □

Згідно з твердженням 2.1, усі НД, що знаходить UID-метод, зможе знайти і метод У-Ваня. Але все ж існує великий клас НД, які можна отримати тільки методом У-Ваня. Для прикладу UID-метод не зможе знайти НД зображений на рисунку

Отже, можна довести, що метод У-Ваня узагальнює в собі UID-метод та U-метод, як часткові випадки. Зокрема, було показано, що за допомогою методу У-Ваня реально знайти неможливі диференціали з більшим значенням довжини.

Важливо також зазначити, що в UID-методі є ще одна знакова відмінність, що відрізняє його від U-методу та власне методу У-Ваня. А саме, в ньому розглядаються відношення між вхідними та вихідними змінними і провіряє чи вони однакові, в той час як інші методи з перечислених використовують тільки поняття нульових та ненульових змінних, оминаючи власне відношення між самими змінними.

2.1

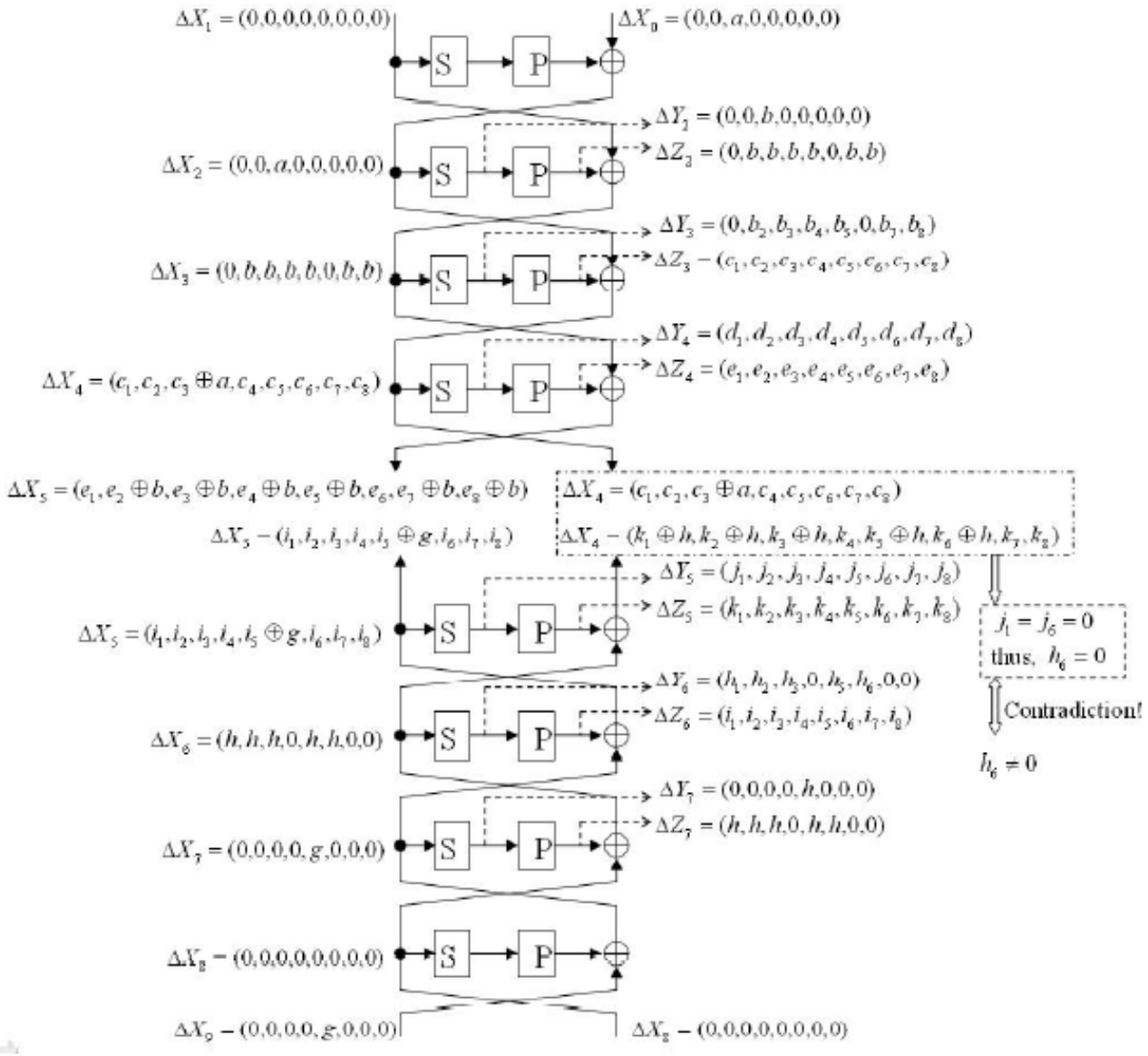


Рисунок 2.1 – Приклад НД для 8-раундового шифру $MIBS$

2.4 Шифр «Калина-2»

До 2015 року ГОСТ 28147-89 був основним блоковим шифром, що використовувався в Україні. Навіть зараз цей шифр як і раніше забезпечує прийнятний рівень стійкості. Проте його реалізація програмного забезпечення є значно повільнішою і менш ефективною на сучасних платформах порівняно з новими рішеннями наприклад в стандарті AES. Шифр «Калина-2» – це криптографічний алгоритм симетричного блокового шифрування. Побудований на основі SP-мережі. Алгоритм був розроблений вітчизняними спеціалістами та був прийнятий у якості національного стандарту України ДСТУ 7624:2014. Це сучасний алгоритм для забезпечення таких задач криптографії як конфіденційність та цілісність.

Для стандарту шифрування визначено 10 режимів роботи алгоритму. Ми будемо розглядати тільки базове перетворення або інакше – режим простої заміни. У такому режимі використання алгоритму для забезпечення конфіденційності не рекомендується. Задля цього в інших режимах є додаткові перетворення, які підвищують криптографічну складність.

В залежності від розмірів блоку та ключа шифр може мати різну кількість ітерації. Ця залежність наведена у таблиці 2.1.

Таблиця 2.1 – Залежність кількості ітерацій від розміру блоку и довжини ключа

Розмір блоку	Довжина ключа	Кількість ітерацій
128	128	10
	256	14
256	256	14
	512	18
512	512	18

Вхідний, вихідний блоки та внутрішній стан перетворення представляється у вигляді матриці розміром $8 * C$ байтів. Кожен байт ми представляємо як елемент поля $GF(2^8)$.

Базове перетворення виконує обробку вхідного блоку даних довжиною l бітів. Матриця внутрішнього стану позначається як $G = (g_{i,j}), g_{i,j} \in D(2^8)$, де $i = 0..7$.

Записуються та зчитуються байти матриці по стовпцях.

Зашифрування. Пряме перетворення алгоритму Калина-2 складається із наступних операцій:

– Функція додавання раундового ключа за модулем 2^{64}.

Цикловий ключ представлений як матриця розміром $8 \cdot C$. Ми додаємо по стовпцям раундовий ключ до стовпців стану перетворення.

– Шар нелінійного бієктивного відображення (байтові підстановки).

На даному етапі ми застосовуємо до кожного байту матриці внутрішнього стану $G = (g_{i,j})$ підстановку $S_k : V_8 \to V_8, k = 0, 1, 2, 3$, де S_k підстановки наведені у ДСТУ.

– Перестановка елементів

Дана операція виконує циклічний зсув вправо рядків матриці стану. Кількість елементів зсуву залежить від номеру рядка та розміру блоку. Обчислюється за формулою $\delta_i = \lfloor i * l/512 \rfloor$. Наприклад, п'ятий рядок станової матриці шифру з 256-бітним розміром блоку циркулярно зміщений вправо на дві позиції.

– Лінійне перетворення

Дана операція виконується наступним чином: кожен елемент матриці стану представляється як елемент $GF(2^8)$, що утворено незвідним поліномом $f(x) = x^8 + x^4 + x^3 + x^2 + 1$. Кожен елемент результуючої матриці стану отримується, як результат множення векторів довжини 8 над полем $GF(2^8)$.

– Функція додавання раундового ключа за модулем 2.

Раундовий ключ в нас представлений, як матриця розміру $8 \cdot c$. Під час цієї операції ми беремо і додаємо байти раундового ключа та байти матриці.

Таблиця 2.2 – Порядок операцій для шифру Калина-2

# ітерації	Операції
0	Додавання раундового ключа за модулем 2^{64}
$1 - -t - 1$	Шар нелінійного бієктивного відображення Перестановка елементів Лінійне перетворення Функція додавання циклового ключа за модулем 2
t	Шар нелінійного бієктивного відображення Перестановка елементів Лінійне перетворення Функція додавання циклового ключа за модулем 2^{64}

У таблиці 2.2 приведено порядок операції у алгоритмі Калина-2.

Отже, «Калина-2» - це блоковий шифр із SPN-подібною структурою. Він має збільшений розмір MDS-матриці, новий набір із чотирьох різних S-боксів і тд. Калина підтримує розмір блоку та довжину ключа в 128, 256 і 512 біт.

2.5 Формалізація Калина-подібних шифрів для уточнення методу У-Ваня

Розібравшись з основними деталями та перетвореннями шифру «Калина-2» та з загальними деталями власне побудови формальних правил для пошуку неможливих диференціалів можемо побачити, що «Калина»-подібні шифри досить зручно виражаються в термінах методу.

Окрім того слід взяти до уваги наступний фактор. Як було показано раніше, система в загальному випадку є неподільною і з ростом довжини НД, росла і складність рішення цієї системи. Кількість комбінацій для пропускання правил сильно росте при великих значеннях. Однак для SP-мереж всі лінійні перетворення розмежовані між собою нелінійними

шарами, а тому лінійні рівняння різних раундів не містять спільних змінних. У AES-подібних та «Калина»-подібних шифрів це твердження справедливо взагалі для кожного окремого стовпчика матриці стану. Відповідно, система лінійних рівнянь розпадається на окремі незалежні підсистеми малого розміру, які треба перевіряти на розв'язуваність.

Як визначав Клод Шеннон у своїй роботі "Communication Theory of Secrecy Communication розсіювання та перемішування є двома властивостями, необхідними для роботи будь-якого захищеного криптографічного алгоритму [1]. У класичних шифрах ці атрибути були досягнуті за допомогою використання шифрів заміни чи перестановки . Сучасна криптографія використовує одне і те ж явище у формі SPN мереж. Фейстелівські мережі для схожих цілей використовують S-блоки.

Дифузія в SPN пов'язана з практикою перестановки блоків, що розсіюють статистичну структуру відкритого тексту в загальній статистиці шифртексту. Максимально розділені по відстані матриці (Maximum Distance Separable (MDS)), які в основному походять від ReedSolomon кодів, доставляють дифузійні властивості, що робить їх однією з життєво важливих складових сучасних шифрів типу AES і «Калина-2».

В MDS-матриць є багато корисних властивостей, які позволяють їх на повну використовувати при побудові шифрів. Одною з них вляється те, що у MDS-матриці всі квадратні підматриці є невиродженими. З цього випливає, що якщо кількість ненульових слів на вході та виході MDS-перетворення не менша за його індекс розгалуження, то відповідна система лінійних рівнянь завжди буде мати розв'язок. Відповідно, якщо система лінійних рівнянь утворена MDS-перетворенням, то перевірка її розв'язуваності фактично зводиться до підрахунку кількості ненульових слів у вхідному та вихідному векторах змінних.

Метод У-Ваня перевіряє сумісність системи лінійних рівнянь лише на самому початку аналізу. Однак під час пошуку суперечностей і встановлення значень проміжних змінних у лінійних рівняннях також

відбуваються зміни, які можуть дати певну суперечність. Перевірка цих умов може надати додаткову інформацію про проходження диференціалу у шифрі та, відповідно, про наявність певних суперечностей.

Окрім доданих в минулій роботі B-співвідношення, що пов'язують вхідні та вихідні змінні кожного MDS-перетворення у шифрі, вводяться новий клас співвідношень для перевірки розв'язуваності системи для кожного з проміжних значень змінних. Ми називатимемо їх C-співвідношеннями і вони також будуть давати додаткову інформацію для отримання НД. Але все ж варто зазначити,щ ця інформація може бути застосовною тільки для отримання НД на цьому етапі. Для наступних же етапів ніякої нової інформації з цих перетворень ми не отримуємо.

Послідовно розглядаючи кроки цієї схеми шифрування можна помітити, що додавання ключів за певним модулем не буде впливати на вигляд диференціалів, тому вони й не розглядатимуться. Таким чином, дійсний вплив на результат матимуть шар нелінійного відображення (який і утворюватиме NL-підсистему), шар перестановки елементів та шар лінійного перетворення, на основі якого можна побудувати матрицю L.

Матриця L будується на основі циркулятивної матриці, яка і задає всі впливові переходи змінних. Циркулятивна матриця в Калині має вигляд вектора $v = (0x01, 0x01, 0x05, 0x01, 0x08, 0x06, 0x07, 0x04)$, що складається з послідовності байтових констант у шіснадцятковому поданні. Всі вони інтерпретуються як елементи поля $GF(2^8)$, при цьому циклічний зсув виконується відносно елементів вектора над скінченним полем.

NL-шар для «Калина»-подібних шифрів принципово не відрізняється від NL-шарів для більшості поширених блокових шифрів. Він також задається набором правил переходів змінних одна в одну і допомагає шукати протиріччя.

Висновки до розділу 2

Даний розділ являється дуже важливим для побудови вже власне готового методу в наступній главі. В цьому розділі на відміну від першого, вже описана власне ідейна та математична база, необхідні для розуміння всіх деталей, що висуватимуться надалі. Оскільки дана робота трактується як продовження моїх попередніх досліджень, то багато описових частин були запозичені з неї. Їх формулювання являється доволі чітким і тому структура даного розділу є аналогічною.

В наступному розділі на основі даного матеріалу вже буде чітко описаний алгоритм для отримання результатів.

3 ВИСУНУТИЙ АЛГОРИТМ ТА ЙОГО РЕАЛІЗАЦІЯ

В даному розділі використовуються всі відомості з попереднього розділу для побудови ефективного методу пошуку неможливих диференціалів. Також вводиться спроба доповнення попередньої моделі новими правилами або ж етапами в алгоритмі для спроби покращення результатів. Ну і як результат, буде приведено чи покращення програмної реалізації згідно з доповненнями до минулого методу дозволяє найти більше неможливих диференціалів.

3.1 Введення змінних на кожному раунді та правила їх обробки

Як вже було описано в попередньому розділі, чимала доля потенційно важливої для знаходження неможливого диференціалу інформації втрачається за рахунок недосконалої моделі типізації змінних. Тобто в попередній реалізації $P[.]$ та $C[.] \subset \{0, 1, 2\}$, де $P[.]$ та $C[.]$ - вхідний та вихідний масиви, і приймають значення нульової, ненульової та невідомої різниці відповідно до значень.

Натомість, ми можемо використати модель в якій значення типу, що рівне 1 натомість варіюватиметься можливими буковними позначеннями в залежності від відношень між власне змінними. Таким чином в цьому методі можливо буде знайти більше можливих комбінацій НД. Для прикладу, якщо $\Delta P = (a, 0, 0, a)$ і $\Delta = (a, 0, 0, b)$, де a та b - різні ненульові значення, то в такому разі можна стверджувати, що $x_1 = x_4 = y_1$ та $y_4 \neq x_1$.

Слід зауважити, що при роботі з системою лінійних рівнянь для

виявлення її сумісності, значення змінних розглядатиметься в контексті нульових та ненульових значень. Диференціювання на відмінні вже між собою значення змінних нас не цікавитиме на даному етапі.

В такому разі виникає ряд запитань, щодо доцільності даного методу в контексті його ефективності. Іншими словами, якщо ми маємо множину можливих значень: $\{0, a_1, ..., a_n, b_1, ..., b_n, 2\}$, де символи $a_i, b_i, 1 \leq i \leq n-2n$ різних точно відмінних від нуля значень. В такоми випадку ми отримаємо $((n+1)^n - 1) \cdot ((2n+1)^n - 1)$ диференційних пар. Для класів шифрів, що ми будемо розглядати, це значення являється занадто великим.

Тим не менш, відштовхуючись від найбільш від результатів минулої роботи і практики використання даного підходу, досить раціональним підходом буде використовувати прості шаблони деференційних пар. Тимпаче, всі НД знайдені для блокових шифрів опубліковані в літературі є простого виду.

Отже, в даній роботі для вхідного та вихідного диференціалу виду $(\Delta P_1, ..., \Delta P_n)$ та , матимуть $\Delta P_i, \Delta P_i \subset \{0, a, b\}$. Таким чином кількість диференційних пар, що розглядатимуться сягатиме $(3^n - 1) \cdot (3^n - 1)$.

З іншої сторони, як і раніше, вибір множини вхідних та вихідних масивів змінних зручно зупинити на масивах виду: $\{x_1, x_2, ..., x_s, ..., x_n\}$, де $x_s \neq 0$, $x_i = 0$, для $i \neq s$.

Підкреслимо ще раз, що змінна відповідає різниці певного машинного слова, адже розглядати в якості змінної різниці бітів дуже затратно та не практично.

Вцілому, вигляд основного циклу в контексті внесених доповнень зовсім не змінюється. Відповідно ініціалізація змінних та правил переходів буде все тією ж.

Основний цикл:

1) Далі необхідно ввести нові змінні, що відповідатимуть можливим значенням масиву після проходження через нелінійний шар, які відповідають

2) Додаємо нелінійні правила переходів через NL-частину виду

(x_i, z_i), де z_i - змінна, в яку переходить x_i при проході через NL-частину.

3) Вводимо нові змінні для описання переходів через лінійну частину L

4) Додаємо відповідні лінійні правила переходів.

Основний цикл ми проганяємо r-раз, де r – кількість раундів.

3.2 Наглядний приклад відмінності нової моделі присвоєнь значень змінним

Давайте розглянемо приклад 5-раундового шифру Фейстелівського типу. Для початку присвоємо диференційні змінні для 5-раундового шифру Фейстелівського типу.

На рисунку 3.1 $F_i, 1 \leq i \leq 5$ є перемішуваннями, де вихідна різниця F_i отримує на вхід X_i і видає на вихід Y_i. Отже позначимо це як $X_i \sim Y_i$.

Згідно з цим же обчислювальним графіком для 5-раундового шифру Фейстелівського типу, ми отримуємо наступну систему рівнянь:

$$\begin{aligned}
&X_0 \oplus X_2 \oplus Y_1 = 0, && X_1 \sim Y_1\\
&X_1 \oplus X_3 \oplus Y_2 = 0, && X_2 \sim Y_2\\
&X_2 \oplus X_4 \oplus Y_3 = 0, && X_3 \sim Y_3\\
&X_3 \oplus X_5 \oplus Y_4 = 0, && X_4 \sim Y_4\\
&X_4 \oplus X_6 \oplus Y_5 = 0, && X_5 \sim Y_5
\end{aligned}$$

Для того, щоб перевірити чи являється $(a, 0) \to (a, 0)$ неможливим диференціалом, де a- ненульового значення потрібно вирішити систему з $X_0 = a, X_1 = 0, X_5 = 0, X_6 = a$. Так як $X_1 \sim Y_1$ та $X_5 \sim Y_5$ повинно бути $Y_1 = 0$ та $Y_5 = 0$. З лінійних правил ми маємо $Y_3 = 0$, а отже і $X_3 = 0$, оскільки $X_3 \sim Y_3$. Далі ж з лінійних рівнянь ми отримуємо, що $Y_2 = 0$, хоча і $X_2 = a$ і $X_2 \sim Y_2$. Таким чином система немає розв'язків і $(a, 0) \to (a, 0)$ є неможливим диференціалом для даного шифру.

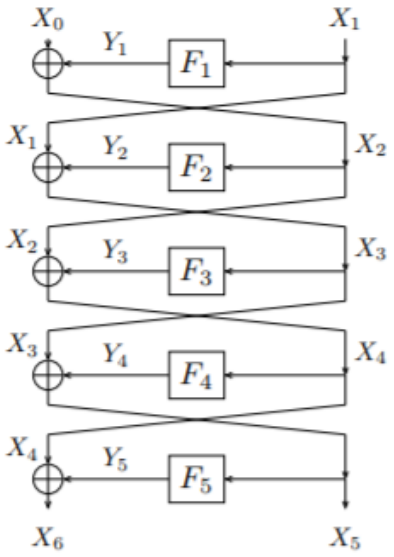


Рисунок 3.1 – Найменування проміжних значень змінних для 5-раундового шифру Фейстелівського типу

Тепер же щоб знайти всі неможливі диференціали для даного шифру, ми перечислюємо всі можливі пари диференційних пар:

$\{(0,a),(a,0),(a,a),(0,b),(b,0),(b,b),(a,b),(b,a)\}$, де a,b мають два відмінні ненульові значення. Прогнавши всі пари диференціалів через аналогічні процедури, ми отримаємо всі неможливі диференціали даного шифру в рамках даної моделі.

Звісно, даний алгоритм не носить в собі узагальнений характер. Він більше схожий на спеціалізований та адаптований під шифри Фейстелівського типу спосіб пошуку неможливих диференціалів. Але все ж він допомагає наглядно розібратись в деяких деталях нового способу введень проміжних значень для змінних.

В основному ці зміни допоможуть ввести деякі нові формальні правила для пропускання змінних через внутрішні примітиви та, звісно, для отримання більш широкого спектру протиріч.

3.3 Схема алгоритму для знаходження неможливих диференціалів максимальної довжини для Калина-подібних шифрів

Нам необхідно, задаючи вибрані нами пари масивів типів змінних входу і виходу шифрів ($P[.]$ та $C[.]$), отримати результат чи являються вони неможливим диференціалом. Тому алгоритм здобуває все нову інформацію з лінійної та нелінійної частини до тих пір, поки не знайдено протиріччя, або ж він ще може здобувати цю нову інформацію.

Схема його роботи виглядає наступним чином:

1) Вибираємо множину вхідних та вихідних масивів типів змінних.

2) Будуємо загальну систему всіх лінійних рівнянь і перевіряємо її на розв'язуваність.

3) Пробігаємо всі NL-правила.

– Якщо знаходимо одночасно нульові та ненульові змінні, то існує неможливий диференціал

– Якщо знаходимо змінні, що одночасно відповідає різним ненульовим значенням, то існує неможливий диференціал даної довжини

– Якщо дві умови не виконуються, то змінюємо значення змінних відносно правил

4) Для L-частини перевіряємо чи існують рівняння виду:

– $a \cdot x = const$ тоді, якщо $const = 0$, $x = 0$, або ж якщо $const \neq 0$, $x \neq 0$.

– $a \cdot x \oplus b \cdot y = const$.

– $0 = const \neq 0$, якщо так – протиріччя.

5) Обробка B правил. Для кожного B-співвідношення [y, z] в описі шифру перевіряємо:

– якщо всі змінні вектору y є нульовими, то зробити всі змінні вектору z нульовими аналогічно і навпаки.

– якщо загальна кількість нульових змінних у векторах y та z є строго більшою за $l-1$, то знайдено протиріччя.

6) Обробка правил. Для кожного C-співвідношення [y, z] в описі шифру перевіряємо чи задана векторами y та z підматриці будуть максимального рангу. Іншими словами це являється провіркою на сумісність лінійної системи рівнянь, яка утворюєтья в даному випадку. Якщо ранг є меншим за максимальний, то знайдено протиріччя.

7) Якщо після обробки всіх правил $X[.]$ не змінився, то завершуємо роботу, адже ми вже не можемо витягнути ніякої додаткової інформації.

Пояснимо більш детально обробку B-правил. Для MDS-перетворення, яке визначається матрицею розміру $l \cdot l$, загальна кількість ненульових змінних на вході та виході є щонайменше $l+1$. Відповідно, загальна кількість ненульових змінних на виході не може перевищувати $l-1$. Таким чином, якщо виявиться, що загальна кількість нульових змінних буде більшою за $l-1$, то це гарантована суперечність у шляху проходження диференціалу шифру.

Слід зазначити, що умови перевірки для лінійної частини типу: $a \cdot x \oplus b \cdot y = const$ та $0 = const \neq 0$, будуть розглядатись як можливе доповнення до запропонованих умов У-Ванем.

Зауваження. Оригінальний шифр «Калина» на початку та наприкінці шифрування використовує додавання із ключем за модулем 2^{64}. Ця операція не є блоковою та в загальному випадку не зберігає структуру шаблонів. Однак шаблони виду, що розглядається в роботі будуть зберігатись при проходженні через ключовий суматор такого виду.

Таким чином, запропонована модель застосовна і до оригінального шифру «Калина» із не блоковими операціями у ключовому суматорі першого та останнього раундів, якщо обмежитись усіченими диференціалами описаного виду.

3.4 Деякі деталі програмної реалізації алгоритму

– За основу програмного ядра бралась програмна реалізація з попередньої роботи

– Враховуючи архітектуру програмного коду, що був написаний для попередньої версії програмного ядра, в більшій мірі модифікація вплинула на дописання частини для роботи з СЛАР та розширенням варіативності можливих значень проміжних змінних.

– Множину можливих пар диференціалів на вхід-вихід, що розглядались при модифікуванні програмної реалізації обмежувалось згідно з принципом,що описаний в секції 3.2.

– Після генерації всіх можливих шаблонів змінних, ініціалізовувались по порядку лінійні та нелінійні правила переходу змінних;

– В лінійних правилах принциповим є кількість змінних в блоці, адже від цього залежить вид лінійної системи, яка в них будується;

– Після цього відбувається поступовий перебір всіх можливих комбінацій згідно з множиною шаблонів і, власне, робота алгоритму з ними;

– Обробка правил ведеться з двох сторін – «знизу» і «зверху» аж до моменту поки вони не перетнуться;

– Власне з цього моменту алгоритм перевіряє на протиріччя дані, що отримав з двох потоків. Він або знаходить його, або ж дістає нову інформацію для обох потоків, згідно з правил обробки для лінійних та нелінійних правил відповідно;

– Якщо наступає момент, коли потік отримав всі змінні невідомого типу, цей потік припиняє роботу;

– Якщо два потоки припинили свою роботу, або ж протиріччя не знайдено на всіх раундах, алгоритм дає негативний результат.

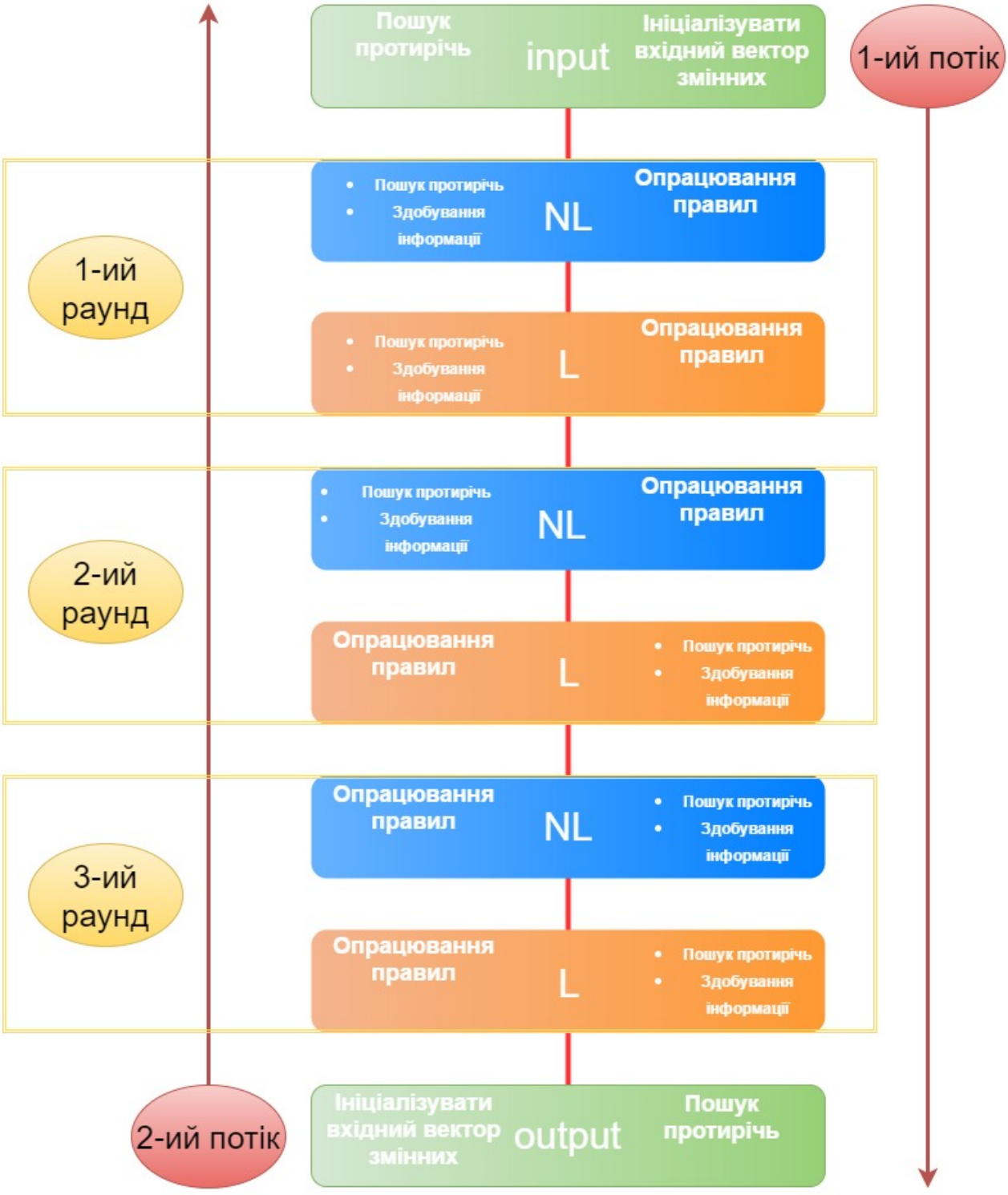


Рисунок 3.2 – Схема роботи алгоритму з «Калина»-подібними шифрами

3.5 Результати експериментальних даних

В попередній роботі в даному напрямку, хоч і метод формалізувався для роботи з широким класом «Калина»-подібних шифрів, але результати були застосовані по більшій мірі до зменшеної версії шифру «Калина» із блоком у 64 біти та до оригінальних версії шифрів «Калина-128» та «Калина-256».

Цього ж разу, враховуючи додані нові правила, була проведена спроба отримати певні порівняльні дані для «Калина»-подібних шифрів з різними матрицями взятими для опрацювання на етапі С-перетворень.

Були розглянуті матриці оригінальної «Калина-2» та її зменшеної версії, як і раніше. Окрім цього були розглянуті матриці шифрів Midori та Camellia. Іншими словами за мету було взято порівняти стійкість даного типу шифрів до відповідних атак в залежності від заміни матриць.

Також на меті дослідження, було дослідити чи реально буде розширити кількість НД або ж збільшити максимальну їх довжину, якщо модифікувати його під отримання більшої кількості проміжної інформації.

Як показали результати, набір НД, які було отримано за допомогою оновленої програмної реалізації залишався аналогічним. Це в свою чергу значить, що весь даний клас шифрів є стійким до всіх покращень які вносились для даного методу. Очікуваного диференціалу довжини 4 з набору розгляданих пар вхід-вихід так і не було отримано.

Таким чином для більшої наглядності сумарних результатів, що були отримані за допомогою даного методу, будуть продубльовані попередні результати в їхній системі позначень.

Для першого шифру було знайдено 900 класів неможливих диференціалів. У всіх диференціалів виникала суперечність під час аналізу лінійних правил і знаходилась за дві ітерації обробки правил.

Наведемо приклад диференціального шляху трираундового неможливого диференціалу. Тут 0 – нульова змінна, 1 – ненульова змінна, 2 – невідома змінна, INPUT – вхідний вектор різниць. На червоному тлі виділено місце виникнення суперечності.

	INPUT:	0000	0001
1-ий раунд	SB:	0000	0001
	SR:	0001	0000
	MC:	1111	0000
2-ий раунд	SB:	1111	0000
	SR:	1100	0011
	MC:	2200	0022
3-ий раунд	SB:	2200	0022
	SR:	2222	0000
	MC:	0001	0000

Рисунок 3.3 – Приклад НД для зменшеної версії шифру «Калина»

Як було написано раніше, чотирираундових шифрів не вдалось знайти жодного неможливого диференціалу.

– для шифру «Калина-128» було знайдено 5184 класи неможливих диференціалів з 18496, що розглядались;

– для шифру «Калина-128» було знайдено 278784 класи неможливих диференціалів з 278784, що розглядались;

Як і для зменшеної версії, в усіх знайдених неможливих диференціалах суперечність виникала під час обробки лінійних правил і знаходилась в 2 ітерації.

Всі неможливі диференціали, які були знайдені на виході містили хоча б одну повністю нульову колонку.

Нижче наведено приклад диференційних шляхів знайдених неможливих диференціалів:

	INPUT:	0000	0000	0000	0101
1-ий раунд	SB:	0000	0000	0000	0101
	SR:	0000	0101	0000	0000
	MC:	2222	2222	0000	0000
2-ий раунд	SB:	2222	2222	0000	0000
	SR:	2222	0000	0000	2222
	MC:	2222	0000	0000	2222
3-ий раунд	SB:	2222	0000	0000	2222
	SR:	2222	2222	0000	0000
	MC:	0000	0001	0000	0000

Рисунок 3.4 – Калина-128

	INPUT:	1000	0000	0000	0000	0000	0000	0000	0000
1-ий раунд	SB:	1000	0000	0000	0000	0000	0000	0000	0000
	SR:	1000	0000	0000	0000	0000	0000	0000	0000
	MC:	1111	1111	0000	0000	0000	0000	0000	0000
2-ий раунд	SB:	1111	1111	0000	0000	0000	0000	0000	0000
	SR:	1100	0000	0000	0011	0000	1100	0000	0011
	MC:	0022	0000	2200	0000	0000	0022	0000	2200
3-ий раунд	SB:	0022	0000	2200	0000	0000	0022	0000	2200
	SR:	0000	0000	2222	2222	0000	0000	0000	0000
	MC:	0000	0001	0000	0000	0000	0000	0000	0000

Рисунок 3.5 – Калина-256

Висновки до розділу 3

В даному розділі було детально описано всі додаткові частини нового методу пошуку НД на основі попереднього та причини по яким вони вводяться. З врахуванням цих доповнень був описаний уточнений метод з додатковими формальними правилами та оновлена програмна реалізація.

Результатом роботи згаданого оновленого програмного ядра виявилось одержання аналогічного сегменту НД тієї ж довжини. Це означає в свою чергу, що дані типи шифрів є стійкими відносно внесених доповнень та не піддаються їм.

ВИСНОВКИ

В ході даної роботи був проведений аналіз основних опублікованих існуючих методів для пошуку неможливих диференціалів, які мали місце для знаходження вагомих класів неможливих диференціалів. Описавши всі ці алгоритми, був чітко описаний послідовний зв'язок між ними та проведений розріз перетікання ідей та частин одного методу в інший. Було показано, що більшість нових в свій час методів, були отримані як більш загальні способи пошуку НД своїх попередників.

В другому розділі були описані базовий метод для пошуку НД на основі якого і робились всі уточнення та адаптування під «Калина»-подібні шифри. Також були розглянуті важливі поняття і теоретичні дані для уточнення конкретно попереднього алгоритму та вказані основні деталі щодо особливостей шифру «Калина-2».

В завершальному третьому розділі були вже детально формалізовані всі правила методу та описані всі доповнення до цих правил. Була проведена аргументація доцільності введення даних додаткових правил та можливий їх вплив на роботу алгоритму.

За результатами роботи програмної реалізації методу було виявлено стійкість «Калина»-подібних шифрів до покращення даної атаки. Іншими словами, ніякої нової інформації відносно старої версії методу не здобувалося.

З іншої сторони, дані дослідження та програмна реалізація методу можуть послужити в майбутньому для подальшого доповнення та уточнення даного методу, або ж більш глобального пошуку НД для схожих шифрів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Shengbao Wu and Mingsheng Wang. Automatic Search of Truncated Impossible Differentials for Word-Oriented Block Ciphers. — Cryptology ePrint Archive, Report 2012/214. — 2012. — http:// eprint.iacr.org/2012/214.

2. Yiyuan Luo, Zhongming Wu, Xuejia Lai and Guang Gong. A Unified Method for Finding Impossible Differentials of Block Cipher Structures. — Cryptology ePrint Archive, Report 2009/627. — 2009. — http://eprint.iacr.org/2009/627.

3. A New Encryption Standard of Ukraine: The Kalyna Block Cipher. Roman Oliynykov, Ivan Gorbenko, Oleksandr Kazymyrov et al. — 2015. — http://eprint.iacr. org/2015/650.

4. Eli Biham, Nathan Kellery. Cryptanalysis of Reduced Variants of Rijndael. — 2002. — Access mode: http://madchat.fr/crypto/codebreakers/35-ebiham.pdf.

5. Wei Wang and Xiaoyun Wang. Improved Impossible Differential Cryptanalysis of CLEFIA. - Cryptology ePrint Archive, Report 2007/466. — 2007. — http://eprint.iacr.org/2009/627.

6. Wenling Wu, Wentao Zhang, and Dengguo Feng. Impossible Differential Cryptanalysis of ARIA and Camellia. - Cryptology ePrint Archive, Report 2016/188. — 2006. — http:// eprint.iacr.org/2006/350.

7. Yiyuan Luo and Xuejia Lai. Impossible Improvements for Finding Impossible Differentials of Block Cipher Structures. - Cryptology ePrint Archive, Report 2017/1209. — 2017. — https://eprint.iacr.org/2017/1209.

8. Muhammad Yasir Malik and Jong-Seon No. Impossible Improvements for Dynamic MDS Matrices for Substantial Cryptographic Strength. - Cryptology ePrint Archive, Report 2011/177. — 2011. — https://eprint.iacr.org/2011/177.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

Нижче наведені найважливіші частини програмного коду, що використовувався для реалізації запропонованого методу пошуку неможливих диференціалів.

А.1 Клас представлення L-правил

А.1.1 Інтерфейс

```
class CLRules
{
public:
    CLRules(int blockSize, int numberOfRounds) ;

    void initializeRules(std::vector<int> namesOfVariables,
        int numberOfCurrentRound);
    void recoverInfoUpDown(std::vector<CVariable> & arrayOfVariables,
        int numberOfRound);
    void recoverInfoDownUp(std::vector<CVariable> & arrayOfVariables,
        int numberOfRound);
    bool checkContradiction(std::vector<CVariable>
        & arrayOfInputVariables,
        std::vector<CVariable> & arrayOfOutputVariables,
        int numberOfRound);
    bool LynearSystemSolviability(std::vector<int> ZeroRaws,
        int numberOfOututZeros);

private:
```

```
    std::vector<int> cycleShiftForRow(std::vector<int> inputRaw,
        int shiftValue);
    std::vector<std::vector<int>> generateKalynaBlockView
        (std::vector<int> input);

    std::vector<std::vector<int>> m_circulantMatrix;
    std::vector<std::vector<int>> m_variableMatrix;
    int m_numberOfColumns;
    int m_firstRowWithCycleShift;
    int m_numberOfRounds;
};
```

A.1.2 Реалізація класу

```
CLRules::CLRules(int blockSize, int numberOfRounds)
{
    m_circulantMatrix[0] = { 0, 1, 1, 1, 1, 0, 0, 1 };
m_circulantMatrix[1] = { 1, 0, 1, 1, 1, 1, 0, 0 };
m_circulantMatrix[2] = { 1, 1, 0, 1, 0, 1, 1, 0 };
m_circulantMatrix[3] = { 1, 1, 1, 0, 0, 0, 1, 1 };
m_circulantMatrix[4] = { 0, 1, 1, 1, 1, 1, 1, 0 };
m_circulantMatrix[5] = { 1, 0, 1, 1, 0, 1, 1, 1 };
m_circulantMatrix[6] = { 1, 1, 0, 1, 1, 0, 1, 1 };
m_circulantMatrix[7] = { 1, 1, 1, 0, 1, 1, 0, 1 };
    m_numberOfRounds = numberOfRounds;
    switch (blockSize)
    {
        case 128:
```

```
        {
            m_numberOfColumns = 2;
            m_firstRowWithCycleShift = 4;
            break;
        }
        case 256:
        {
            m_numberOfColumns = 4;
            m_firstRowWithCycleShift = 2;
            break;
        }
        case 512:
        {
            m_numberOfColumns = 8;
            m_firstRowWithCycleShift = 1;
            break;
        }
        default:
            m_numberOfColumns = 0;
            break;
    }
    m_variableMatrix = std::vector<std::vector<int>>
        (m_numberOfColumns*m_numberOfRounds,
        std::vector<int>(16, 0));
}

void CLRules::initializeRules(std::vector<int> namesOfVariables,
    int numberOfCurrentRound)
{
    int currentShiftValue = 0;
    int nextRowWithShift = m_firstRowWithCycleShift;
```

```
    std::vector<int> initializedVector(16,0);    // !!!
    auto rawVectors = generateKalynaBlockView(namesOfVariables);
    for (int i = 0; i < 8; ++i)
    {
        if (i < nextRowWithShift)
        {
            rawVectors[i] = cycleShiftForRow(rawVectors[i],
                currentShiftValue);
        }
        else
        {
            nextRowWithShift += m_firstRowWithCycleShift;
            currentShiftValue += 1;
            rawVectors[i] = cycleShiftForRow(rawVectors[i],
                currentShiftValue);
        }
    }
    for (int i = 0; i < m_numberOfColumns; ++i)
    {
        for (int j = 0; j < 8; ++j)
        {
            initializedVector[j] = rawVectors[j][i];
            initializedVector[j + 8] = namesOfVariables[j]
                + namesOfVariables.size() + i*8;
        }
        m_variableMatrix[i + numberOfCurrentRound
            * m_numberOfColumns] = initializedVector;
    }
}

void CLRules::recoverInfoUpDown(std::vector<CVariable>&
```

```
    arrayOfVariables, int numberOfRound)
{
    for (int i = 0; i < m_numberOfColumns; ++i)
    {
        auto tmpColumn = m_variableMatrix[i +
            numberOfRound*m_numberOfColumns];
        for (int j = 0; j < 8; ++j)
        {
            bool existSingleNonzeroVar = false;
            bool existUnknownVar = false;
            bool allVarsAreZero = true;
            for (int k = 0; k < 8; ++k)
            {
                if (arrayOfVariables
                    [tmpColumn[k]].getType() == 1)
                {
                    if ((!existSingleNonzeroVar))
                    {
                        existSingleNonzeroVar = true;
                    }
                    else
                    {
                        existUnknownVar = true;
                        break;
                    }
                    allVarsAreZero = false;
                }
                else if (arrayOfVariables
                    [tmpColumn[k]].getType() == 2)
                {
                    existUnknownVar = true;
```

```
                    break;
                }
            }
            if (existUnknownVar)
            {
                for (int k = 0; k < 8; ++k)
                {
                    arrayOfVariables[tmpColumn[k + 8]].setType(2);
                }
                break;
            }
            if (existSingleNonzeroVar)
            {
                arrayOfVariables[tmpColumn[j + 8]].setType(1);
            }
            if (allVarsAreZero)
            {
                arrayOfVariables[tmpColumn[j + 8]].setType(0);
            }
        }
    }
}

void CLRules::recoverInfoDownUp
    (std::vector<CVariable>& arrayOfVariables, int numberOfRound)
{
    for (int i = 0; i < m_numberOfColumns; ++i)
    {
        auto tmpColumn = m_variableMatrix[i +
            (m_numberOfRounds - numberOfRound - 1)*m_numberOfColumns];
        int numberOfZeroVars = 0;
```

```
        int numberOfNonzeroVars = 0;
        for (int j = 0; j < 8; ++j)
        {
            if (arrayOfVariables[tmpColumn[j + 8]].getType() != 0)
            {
                for (int k = 0; k < 8; ++k)
                {
                    arrayOfVariables[tmpColumn[k]].setType(2);
                }
                break;
            }
        }

    }
}

bool CLRules::checkContradiction
    (std::vector<CVariable> & arrayOfInputVariables,
    std::vector<CVariable> & arrayOfOutputVariables,
    int numberOfRound)
{
    for (int i = 0; i < m_numberOfColumns; ++i)
    {
        auto tmpColumn = m_variableMatrix[i +
            numberOfRound*m_numberOfColumns];
        std::vector<int> indexesOfNonzeroVars(8, 0);
        std::vector<int> indexesOfZeroRaws(8, 0);
std::vector<int> indexesOfZeroColumns(8, 0);
        for (int j = 0; j < 8; ++j)
        {
            int numberOfInputZeroVars = 0;
```

```
for (int k = 0; k < 8; ++k)
{
    if (arrayOfInputVariables
        [tmpColumn[k]].getType() == 0)
    {
        ++numberOfInputZeroVars;
        indexesOfZeroColumns[k] = 1;
    }
    else
    {
        indexesOfNonzeroVars[k - numberOfInputZeroVars]
            = k + i * 8;
    }
}
if (8 - numberOfInputZeroVars < 2)
{
    if (8 - numberOfInputZeroVars == 1)
    {
        if ((arrayOfInputVariables[tmpColumn
            [indexesOfNonzeroVars[0]]].getType() == 1) &&
            (arrayOfOutputVariables[tmpColumn
            [j + 8]].getType() == 0))
        {
            return true;
        }
        else if ((arrayOfInputVariables[tmpColumn
            [indexesOfNonzeroVars[0]]].getType() == 0) &&
            (arrayOfOutputVariables[tmpColumn
            [j + 8]].getType() == 1))
        {
            return true;
```

```
            }
        }
        else if (8 - numberOfInputZeroVars == 2)
        {
            if ((arrayOfOutputVariables[tmpColumn
                [j + 8]].getType() == 0) &&
                (arrayOfInputVariables[tmpColumn
                [indexesOfNonzeroVars[0]]].getType() == 1) &&
                (arrayOfInputVariables[tmpColumn
                [indexesOfNonzeroVars[1]]].getType() == 2))
            {
                arrayOfInputVariables[tmpColumn
                    [indexesOfNonzeroVars[1]]].setType(1);
            }
            else if ((arrayOfOutputVariables[tmpColumn
                [j + 8]].getType() == 0) &&
                (arrayOfInputVariables[tmpColumn
                [indexesOfNonzeroVars[0]]].getType() == 2) &&
                (arrayOfInputVariables[tmpColumn
                [indexesOfNonzeroVars[1]]].getType() == 1))
            {
                arrayOfInputVariables[tmpColumn
                [indexesOfNonzeroVars[0]]].setType(1);
            }
        }
    }
    else
    {
        int numberOfOutputZeroVars = 0;
        for (int k = 0; k < 8; ++k)
        {
```

```
                    if (arrayOfOutputVariables[tmpColumn
                        [k + 8]].getType() == 0)
                    {
                        ++numberOfOutputZeroVars;
                        indexesOfZeroRaws[k] = 1;
                    }
                }
                if (numberOfInputZeroVars
                    + numberOfOutputZeroVars > 7)
                {
                    return true;
                }
                if (LynearSystemSolviability(indexesOfZeroRaws,
{
return true;
}
            }
        }
    }
    return false;
}

std::vector<int> CLRules::cycleShiftForRow(std::vector<int> inputRaw,
     int shiftValue)
{
    std::vector<int> outputRaw(inputRaw.size(),0);
    for (int i = 0; i < inputRaw.size(); ++i)
    {
        outputRaw[i] = inputRaw[(i - shiftValue) % inputRaw.size()];
    }
    return outputRaw;
```

```
}

std::vector<std::vector<int>> CLRules::generateKalynaBlockView
    (std::vector<int> input)
{
    std::vector<std::vector<int>> resultVector
        (8 ,std::vector<int>(m_numberOfColumns, 0));
    std::vector<int> tmpVector(input.size() / m_numberOfColumns, 0);
    for (int i = 0; i < 8; ++i)
    {
        for (int j = 0; j < m_numberOfColumns; ++j)
        {
            resultVector[i][j] = input[i+8*j];
        }
    }
    return resultVector;
}

void swap(std::vector<std::vector<int>> & matrix, int row1, int row2,
int col)
{
for (int i = 0; i < col; i++)
{
int temp = matrix[i][row1];
matrix[i][row1] = matrix[i][row2];
matrix[i][row2] = temp;
}
}

int calcRankOfMatrix(std::vector<std::vector<int>> & matrix)
{
```

```
int rank = matrix.size();

for (int row = 0; row < rank; row++)
{
if (matrix[row][row])
{
for (int col = 0; col < matrix[0].size(); col++)
{
if (col != row)
{
double mult = (double)matrix[row][col] /
matrix[row][row];
for (int i = 0; i < rank; i++)
matrix[i][col] -= mult * matrix[i][row];
}
}
}
else
{
bool reduce = true;

for (int i = row + 1; i < matrix[0].size(); i++)     //
{
if (matrix[row][i])
{
swap(matrix, row, i, rank);
reduce = false;
break;
}
}
```

```
if (reduce)
{
rank--;

for (int i = 0; i < matrix[0].size(); i++)
matrix[row][i] = matrix[rank][i];
}

row--;
}

}
return rank;
}

bool CLRules::LynearSystemSolviability(std::vector<int> zeroRaws, std:
int numberOfInputZeros, int numberOfOutputZeros)
{
std::vector<std::vector<int>> reducedMatrix(8 - numberOfOutputZeros,
std::vector<int>(8 - numberOfInputZeros, 0));
int reducedMatrixNumOfRaw = 0;
for (int i = 0; i < zeroRaws.size(); ++i)
{
if (zeroRaws[i] == 1)
continue;
else
{
int reducedMatrixNumOfColumn = 0;
for (int j = 0; j < zeroColumns.size(); ++j)
{
if (zeroColumns[j] == 1)
```

```
continue;
else
{
reducedMatrix[reducedMatrixNumOfRaw][reducedMatrixNumOfColumn]
= m_circulantMatrix[i][j];
}
++reducedMatrixNumOfColumn;
}
++reducedMatrixNumOfRaw;
}
}
int p2 = calcRankOfMatrix(reducedMatrix);
return (calcRankOfMatrix(reducedMatrix) < numberOfOutputZeros);
}
```

A.2 Клас представлення NL-правил

A.2.1 Інтерфейс

```
class CNLRules
{
public:
    CNLRules(int blockSize, int numberOfRounds):
    m_numberOfVariables(blockSize/8),
    m_numberOfRounds(numberOfRounds){};

    void initializeRules(std::vector<int> namesOfVariables,
         int numberOfCurrentRound);
    void recoverInfoUpDown(std::vector<CVariable> & arrayOfVariables,
```

```
        int numberOfRound);
    void recoverInfoDownUp(std::vector<CVariable> & arrayOfVariables,
        int numberOfRound);

    bool checkContradiction
        (std::vector<CVariable> & arrayOfInputVariables,
        std::vector<CVariable> & arrayOfOutputVariables,
        int numberOfRound);

private:
    int m_numberOfVariables;
    int m_numberOfRounds;
    std::map<int, std::vector<int>> correspondingPairsOfNames;
};
```

A.2.2 Реалізація класу NL

```
void CNLRules::initializeRules(std::vector<int> namesOfVariables,
    int numberOfCurrentRound)
{
    std::vector<int> tmpVector1(m_numberOfVariables,0);
    std::vector<int> tmpVector2(m_numberOfVariables, 0);
    for (int i = 0; i < m_numberOfVariables; ++i)
    {
        tmpVector1[i] = namesOfVariables[i];
        tmpVector2[i] = namesOfVariables[i] + m_numberOfVariables;
    }
    correspondingPairsOfNames
```

```
        [2 * numberOfCurrentRound] = tmpVector1;
    correspondingPairsOfNames
        [2 * numberOfCurrentRound + 1] = tmpVector2;
}

void CNLRules::recoverInfoUpDown
    (std::vector<CVariable>& arrayOfVariables, int numberOfRound)
{
    for (int i = 0; i < m_numberOfVariables; ++i)
    {
        if (arrayOfVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]].getType() == 0)
        {
            arrayOfVariables[correspondingPairsOfNames
                [2 * numberOfRound][i]
                + m_numberOfVariables].setType(0);
        }
        if (arrayOfVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]].getType() == 1)
        {
            arrayOfVariables[correspondingPairsOfNames
                [2 * numberOfRound][i]
                + m_numberOfVariables].setType(1);
        }
        if (arrayOfVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]].getType() == 2)
        {
            arrayOfVariables[correspondingPairsOfNames
                [2 * numberOfRound][i]
                + m_numberOfVariables].setType(2);
        }
```

```
    }
}

void CNLRules::recoverInfoDownUp
    (std::vector<CVariable>& arrayOfVariables, int numberOfRound)
{
    for (int i = 0; i < m_numberOfVariables; ++i)
    {
        int r = m_numberOfRounds - numberOfRound - 1;
        if (arrayOfVariables[correspondingPairsOfNames
            [2 * r][i] + m_numberOfVariables].getType() == 0)
        {
            arrayOfVariables[correspondingPairsOfNames
                [2 * r][i]].setType(0);
        }
        if (arrayOfVariables[correspondingPairsOfNames
            [2 * r][i] + m_numberOfVariables].getType() == 1)
        {
            arrayOfVariables[correspondingPairsOfNames
                [2 * r][i]].setType(1);
        }
        if (arrayOfVariables[correspondingPairsOfNames[2 * r][i]
            + m_numberOfVariables].getType() == 2)
        {
            arrayOfVariables[correspondingPairsOfNames
                [2 * r][i]].setType(2);
        }
    }
}

bool CNLRules::checkContradiction
```

```
    (std::vector<CVariable> & arrayOfInputVariables,
    std::vector<CVariable> & arrayOfOutputVariables,
    int numberOfRound)
{
    int numberOfZeroVars = 0;
    for (int i = 0; i < m_numberOfVariables; ++i)
    {
        if ((arrayOfInputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]].getType() == 0) &&
            (arrayOfOutputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]
            + m_numberOfVariables].getType() == 1))
        {
            return false;
        }
        if ((arrayOfInputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]].getType() == 1) &&
            (arrayOfOutputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]
            + m_numberOfVariables].getType() == 0))
        {
            return false;
        }
        if ((arrayOfInputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]].getType() == 1) &&
            (arrayOfOutputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]
            + m_numberOfVariables].getType() == 2))
        {
            arrayOfOutputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i] + m_numberOfVariables].setType(1);
```

```
        }
        if ((arrayOfInputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]].getType() == 2) &&
            (arrayOfOutputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]
            + m_numberOfVariables].getType() == 1))
        {
            arrayOfInputVariables[correspondingPairsOfNames
                [2 * numberOfRound][i]].setType(1);
        }
        if ((arrayOfInputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]].getType() == 0) &&
            (arrayOfOutputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i] +
            m_numberOfVariables].getType() == 2))
        {
            arrayOfOutputVariables[correspondingPairsOfNames
                [2 * numberOfRound][i]
                + m_numberOfVariables].setType(0);
        }
        if ((arrayOfInputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]].getType() == 2) &&
            (arrayOfOutputVariables[correspondingPairsOfNames
            [2 * numberOfRound][i]
            + m_numberOfVariables].getType() == 0))
        {
            arrayOfInputVariables[correspondingPairsOfNames
                [2 * numberOfRound][i]].setType(0);
        }
    }
    return true;
```

```
}
```

A.3 Основна програма

```
void buildPropagationRules(int blockSize, int numberOfRounds,
    CNLRules* NL, CLRules* L,
    std::vector<CVariable> & arrayOfInputVariables,
    std::vector<CVariable> & arrayOfOutputVariables)
{
    int numberOfVariables = blockSize / 8;

    for (int i = 0; i < arrayOfInputVariables.size(); ++i)
    {
        arrayOfInputVariables[i].setName(i);
        arrayOfOutputVariables[i].setName(i);
    }
    std::vector<int> namesOfVariables(numberOfVariables,0);
    for (int i = 0; i < numberOfRounds; ++i)
    {
        for (int j = 2*i*numberOfVariables;
            j < (2*i + 1)*numberOfVariables; ++j)
        {
            namesOfVariables[j % numberOfVariables] = j;
        }
        NL->initializeRules(namesOfVariables,i);
        for (int j = (2*i + 1)*numberOfVariables;
            j < 2*(i + 1)*numberOfVariables; ++j)
        {
```

```
            namesOfVariables[j % numberOfVariables] = j;
        }
        L->initializeRules(namesOfVariables, i);
    }
}

void generatePossibleCombination(int blockSize,
    int numberOfRounds, std::vector<std::vector<int>> & input)
{
    for (int i = 0; i < blockSize / 8; ++i)
    {
        for (int j = 0; j < blockSize / 8; ++j)
        {
            if (j == i)
            {
                input[i][j] = 1;
            }
            else
            {
                input[i][j] = 0;
            }
        }
    }
};

void generatePossibleCombination1(int blockSize,
    int numberOfRounds, std::vector<std::vector<int>> & input)
{
    int lastIndex = 0;
    for (int i = 0; i < blockSize / 8; ++i)
    {
```

```
        for (int j = 0; j < blockSize / 8 - i; ++j)
        {
            for (int k = 0; k < blockSize / 8; ++k)
            {
                input[j + lastIndex][k] = 0;
            }
            input[j + lastIndex][i] = 1;
            input[j + lastIndex][(i + j) % (blockSize / 8)] = 1;
        }
        lastIndex += blockSize / 8 - i;
    }
};

void resetVariablesValues(std::vector<int> & inputValues,
    std::vector<int> & outputValues,
    std::vector<CVariable> & arrayOfInputVariables,
    std::vector<CVariable> & arrayOfOutputVariables)
{
    for (int i = 0; i < inputValues.size(); ++i)
    {
        arrayOfInputVariables[i].setType(inputValues[i]);
        arrayOfOutputVariables[i + arrayOfInputVariables.size()
        - outputValues.size()].setType(outputValues[i]);
    }
    for (int i = inputValues.size();
        i < arrayOfInputVariables.size(); ++i)
    {
        arrayOfInputVariables[i].setType(0);
        arrayOfOutputVariables[i - outputValues.size()].setType(0);
    }
}
```

```
int main()
{
    int blockSize = 256;
    int numberOfRounds = 3;
    CNLRules* NL = new CNLRules(blockSize, numberOfRounds);
    CLRules* L = new CLRules(blockSize, numberOfRounds);

    std::vector<std::vector<int>> consideredBlockVarsTypes
        (((blockSize / 8 - 1) + (blockSize / 8 - 1)
        *(blockSize / 8 - 1)) / 2 + blockSize / 8,

    std::vector<int>(blockSize / 8, 0));

    std::vector<CVariable> arrayOfInputVariables((blockSize/8)
    * (1 + 2 * numberOfRounds), CVariable());
    std::vector<CVariable> arrayOfOutputVariables((blockSize / 8)
    * (1 + 2 * numberOfRounds), CVariable());

    buildPropagationRules(blockSize, numberOfRounds, NL, L,
        arrayOfInputVariables, arrayOfOutputVariables);

    generatePossibleCombination1(blockSize, numberOfRounds,
        consideredBlockVarsTypes);
    int numberOfIDWithThisLength = 0;
    for (int j = 0; j < ((blockSize / 8 - 1)
        + (blockSize / 8 - 1)*(blockSize / 8 - 1))
        / 2 + 16; ++j)
    {
        for (int k = 0; k < ((blockSize / 8 - 1)
```

```
    + (blockSize / 8 - 1)*(blockSize / 8 - 1))
    / 2 + 16; ++k)
{
    resetVariablesValues(consideredBlockVarsTypes[j],
        consideredBlockVarsTypes[k], arrayOfInputVariables,
        arrayOfOutputVariables);
    bool findedND1 = false, findedND2 = false,
        findedND3 = false, findedND4 = false;
    for (int i = 0; i < numberOfRounds; ++i)
    {
        if (!(findedND1 || findedND2
            || findedND3 || findedND4))
        {
            bool oddnumberOfRounds = numberOfRounds % 2;
            bool readyForComparing = false;
            if (!readyForComparing)
            {
                if (2 * i < numberOfRounds - 1)
                {
                    NL->recoverInfoUpDown
                        (arrayOfInputVariables, i);
                    L->recoverInfoUpDown
                        (arrayOfInputVariables, i);
                    L->recoverInfoDownUp
                        (arrayOfOutputVariables, i);
                    NL->recoverInfoDownUp
                        (arrayOfOutputVariables, i);
                }
                else
                {
                    readyForComparing = true;
```

```
        }
    }
    if (readyForComparing)
    {
        if (numberOfRounds - 2 * i > 0)
        {
            if (oddnumberOfRounds)
            {
                NL->recoverInfoUpDown
                    (arrayOfInputVariables, i);
                L->recoverInfoDownUp
                    (arrayOfOutputVariables, i);
                findedND1 = L->checkContradiction
                    (arrayOfInputVariables,
                    arrayOfOutputVariables, i);
            }
            else
            {
                findedND1 = L->checkContradiction
                    (arrayOfInputVariables,
                    arrayOfOutputVariables, i);
                L->recoverInfoUpDown
                    (arrayOfInputVariables, i);
                NL->recoverInfoDownUp
                    (arrayOfOutputVariables, i);
            }
        }
        else
        {
            findedND1 = NL->checkContradiction
                (arrayOfInputVariables,
```

```
                                    arrayOfOutputVariables, i);
                                findedND2 = L->checkContradiction
                                    (arrayOfOutputVariables,
                                    arrayOfOutputVariables, i);
                                findedND3 = L->checkContradiction
                                    (arrayOfInputVariables,
                                    arrayOfOutputVariables, i);
                                findedND4 = NL->checkContradiction
                                    (arrayOfOutputVariables,
                                    arrayOfOutputVariables, i);
                            }
                        }
                    }
                    else
                    {
                        ++numberOfIDWithThisLength;
                    }
                }
            }
        }

        std::cout << numberOfIDWithThisLength;

        std::cin.get();
        std::cin.get();
        return 0;
    }
```